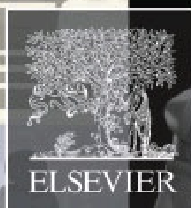
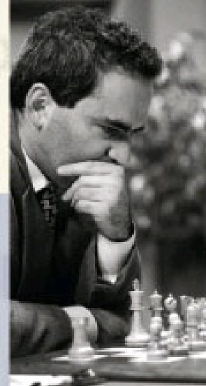




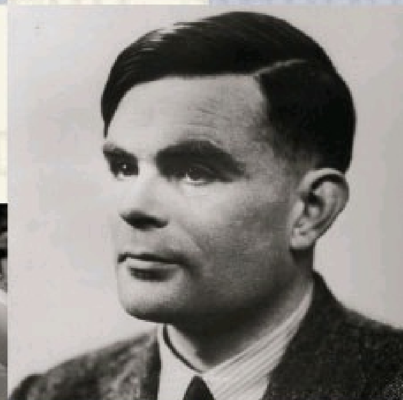












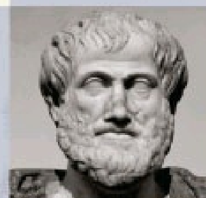













Stuart **Russell**
Peter **Norvig**

Inteligência Artificial

Tradução da Terceira Edição

Referência Completa para Cursos de Computação

Adotado em mais de 750 Universidades em 85 Países



CAMPUS

Stuart
Russell
Peter
Norvig

Inteligência Artificial

Tradução da Terceira Edição

Referência Completa para Cursos de Computação
Adotado em mais de 750 Universidades em 85 Países

Tradução:
Regina Célia Simille de Macedo

Consultoria Editorial e Revisão Técnica:
Flávio Soares Corrêa da Silva, Dr.
Leliane Nunes de Barros, Dra.
Renata Wassermann, Dra.





Do original:
Artificial Intelligence
Tradução autorizada do idioma inglês da edição publicada por Prentice Hall

Coordenação de produção: Silvia Lima

Copidesque: Ivone Teixeira

Editoração eletrônica: DTPhoenix Editorial

Revisão gráfica: Marília Pinto de Oliveira

Conversão para eBook: Freitas Bastos

Elsevier Editora Ltda.
Conhecimento sem Fronteiras
Rua Sete de Setembro, 111 – 16º andar
20050-006 – Centro – Rio de Janeiro – RJ – Brasil

Rua Quintana, 753 – 8º andar
04569-011 – Brooklin – São Paulo – SP – Brasil

Serviço de Atendimento ao Cliente
0800-0265340
sac@elsevier.com.br

ISBN da edição original: 978-0136042594
ISBN: 978-85-352-3701-6

CIP-Brasil. Catalogação-na-fonte.
Sindicato Nacional dos Editores de Livros, RJ

R925i

Russell, Stuart J. (Stuart Jonathan), 1962-
Inteligência artificial / Stuart Russell, Peter Norvig; tradução Regina Célia Simille. – Rio de Janeiro: Elsevier, 2013.

Tradução de: Artificial intelligence, 3rd ed.
Inclui bibliografia e índice
ISBN 978-85-352-3701-6

1. Inteligência artificial. I. Norvig, Peter, 1956- II. Título.

11-5978

CDD: 006.3
CDU: 004.81

Para Loy, Gordon, Lucy e Isaac — S.J.R.

Para Kris, Isabella e Juliet — P.N.

Prefácio

A **inteligência artificial** (IA) é um grande campo, e este é um grande livro. Tentamos explorar toda a extensão do assunto, que abrange lógica, probabilidade e matemática do contínuo, além de percepção, raciocínio, aprendizado, ação e, ainda, tudo o que se refere à eletrônica, desde dispositivos microeletrônicos até robôs para exploração planetária. O livro também é grande porque nos aprofundamos na apresentação de resultados.

O subtítulo deste livro é "Uma Abordagem Moderna". O significado pretendido dessa frase um tanto vazia é que tentamos sintetizar o que hoje é conhecido numa estrutura comum, em vez de tentarmos explicar cada subcampo da IA em seu próprio contexto histórico. Pedimos desculpas àqueles que trabalham em subcampos, que, como resultado, receberam menos reconhecimento do que deveriam.

Novidades desta edição

Esta edição capturou as mudanças em IA que tiveram lugar desde a última edição em 2003. Houve aplicações importantes de tecnologia de IA, tais como a implantação generalizada da prática de reconhecimento de fala, tradução automática, veículos autônomos e robótica de uso doméstico. Houve marcos em algoritmos, como a solução do jogo de damas, e um significativo progresso teórico, particularmente em áreas como a do raciocínio probabilístico, aprendizado de máquina e visão computacional. Mais importante, do nosso ponto de vista, é a evolução contínua na maneira como pensamos sobre essa área e, dessa forma, como organizamos este livro. As principais mudanças foram as seguintes:

- Colocamos mais ênfase em ambientes parcialmente observáveis e não determinísticos, especialmente nas configurações não probabilísticas de pesquisa e planejamento. Os conceitos de *estado de crença* (um conjunto de mundos possíveis) e *estimação de estado* (manutenção do estado de crença) foram introduzidos nesta versão; mais adiante, adicionamos probabilidades.
- Além de discutir os tipos de ambientes e tipos de agentes, agora cobrimos com mais profundidade os tipos de *representações* que um agente pode utilizar. Distinguimos entre representações *atômicas* (em que cada estado do mundo é tratado como uma caixa-preta), representações *fatoradas* (em que um estado é um conjunto de atributos/pares de valor) e representações *estruturadas* (em que o mundo consiste em objetos e relações entre eles).
- Nossa cobertura do planejamento aprofundou-se sobre o planejamento contingente em ambientes parcialmente observáveis, incluindo uma nova abordagem para o planejamento hierárquico.
- Adicionamos um novo material de modelos probabilísticos de primeira ordem, incluindo modelos de *universo aberto* para casos de incerteza quanto à existência de objetos.
- Reescrevemos totalmente o capítulo introdutório de aprendizado de máquina, salientando uma variedade ampla de aprendizagem mais moderna de algoritmos, colocando-os em um patamar

teórico mais consistente.

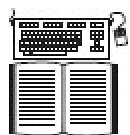
- Expandimos a cobertura de pesquisa na Web e de extração de informações e de técnicas de aprendizado a partir de conjuntos de dados muito grandes.
- 20% das citações desta edição são de trabalhos publicados depois de 2003.
- Estimamos que 20% do material é novo. Os 80% restantes refletem trabalhos mais antigos, mas foram amplamente reescritos para apresentar uma imagem mais unificada da área.

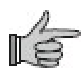
Visão geral do livro

O principal tema unificador é a ideia de **agente inteligente**. Definimos a IA como o estudo de agentes que recebem percepções do ambiente e executam ações. Cada agente implementa uma função que mapeia sequências de percepções em ações, e abordaremos diferentes maneiras de representar essas funções, tais como sistemas de produção, agentes reativos, planejadores condicionais em tempo real, redes neurais e sistemas de teoria de decisão. Explicaremos o papel da aprendizagem como uma extensão do alcance do projetista em ambientes desconhecidos e mostraremos que esse papel restringe o projeto de agentes, favorecendo a representação explícita do conhecimento e do raciocínio. Trataremos da robótica e da visão, não como problemas definidos independentemente, mas como ocorrendo a serviço da realização de objetivos. Enfatizamos a importância do ambiente da tarefa na determinação do projeto apropriado de agentes.

Nosso principal objetivo é transmitir as *ideias* que emergiram nos últimos cinquenta anos de pesquisa sobre a IA e nos dois últimos milênios de trabalhos relacionados a esse tema. Procuramos evitar uma formalidade excessiva na apresentação dessas ideias, ao mesmo tempo em que tentamos preservar a exatidão. Quando consideramos apropriado, incluímos algoritmos em pseudocódigo para tornar as ideias concretas; nosso pseudocódigo é descrito de forma sucinta no Apêndice B.

Este livro se destina principalmente ao uso em cursos de graduação ou de extensão. Também pode ser usado em curso de pós-graduação (talvez com a inclusão de algumas das principais fontes de consulta sugeridas nas notas bibliográficas). O único pré-requisito é a familiaridade com os conceitos básicos de ciência da computação (algoritmos, estruturas de dados, complexidade) em nível básico; os fundamentos matemáticos necessários encontram-se no Apêndice A.


Os exercícios que exigem programação significativa estão marcados com um **ícone de teclado**. Esses exercícios podem ser mais bem resolvidos aproveitando-se o repositório de código em `aima.cs.berkeley.edu`. Alguns deles são grandes o suficiente para serem considerados projetos semestrais. Vários exercícios exigem alguma investigação da literatura de referência; esses exercícios estão marcados com o ícone de **livro**.


Ao longo do livro, os pontos importantes estão indicados por um pequeno ícone de mão *apontando*. Incluímos um índice extenso, com cerca de 6.000 termos, a fim de facilitar a localização de itens no livro. Onde quer que um **novo termo** seja definido pela primeira vez, ele também estará indicado na margem.

Sobre o site*

`aima.cs.berkeley.edu`, **o site do** livro, contém:

- implementações dos algoritmos do livro em várias linguagens de programação
- uma lista de mais de 1.000 escolas que utilizaram o livro, muitas com links para materiais de cursos on-line e ementas
- uma lista comentada com mais de 800 links para diversos sites com conteúdo útil de IA
- uma lista, capítulo por capítulo, de materiais de consulta e links suplementares
- instruções sobre como participar de um grupo de discussão referente ao livro
- instruções sobre como entrar em contato com os autores para fazer perguntas ou comentários

Sobre a capa

A capa mostra a posição final do jogo decisivo da partida 6 de 1997 entre o campeão de xadrez Garry Kasparov e o programa DEEP BLUE. Kasparov, com a cor preta, foi forçado a desistir, tornando essa a primeira vez que um computador derrotou um campeão do mundo em uma partida de xadrez. No topo está a imagem de Kasparov. À sua esquerda está o robô humanoide Asimo e, à sua direita, Thomas Bayes (1702-1761), cujas ideias sobre probabilidade, como medida de crença, formam a base de muito da tecnologia moderna de IA. Abaixo vemos o MarsExploration Rover, um robô que aterrissou em Marte em 2004 e tem explorado o planeta desde então. À direita está Alan Turing (1912-1954), cujo trabalho fundamental definiu os campos da ciência da computação em geral e da inteligência artificial em particular. No fundo está Shakey (1966-1972), o primeiro robô a combinar percepção, modelagem do mundo, planejamento e aprendizado. Junto com Shakey está o líder de projeto Charles Rosen (1917-2002). Embaixo à direita está Aristóteles (384-322 a.C.), pioneiro no estudo da lógica, seu trabalho foi o estado da arte até o século XIX (cópia de um busto por Lisipo). Na parte inferior à esquerda, levemente escondido atrás dos nomes dos autores, está um algoritmo de planejamento por Aristóteles de *De Motu Animalium* no original em grego. Atrás do título está uma porção da rede bayesiana CPSC para diagnóstico médico (Pradhan *et al*., 1994). Atrás do tabuleiro de xadrez encontra-se parte do modelo lógico bayesiano para detectar explosões nucleares a partir de sinais sísmicos.

Créditos: Stan Honda/Getty (Kasparov), Biblioteca do Congresso (Bayes), Nasa (Mars rover), National Museum of Rome (Aristóteles), Peter Norvig (livro), Ian Parker (silhueta de Berkeley), Shutterstock (Asimo, peças de xadrez), Time Life/Getty (Shakey, Turing).

Agradecimentos

Este livro não teria sido possível sem os muitos colaboradores cujos nomes não consegui colocar na capa. Jitendra Malik e David Forsyth escreveram o Capítulo 24 (visão computacional) e Sebastian Thrun escreveu o Capítulo 25 (robótica). Vibhu Mittal escreveu parte do Capítulo 22 (linguagem natural). Nick Hay, Mehran Sahami e Ernest Davis escreveram alguns dos exercícios. Zoran Duric

(George Mason), Thomas C. Henderson (Utah), Leon Reznik (RIT), Michael Gourley (Central Oklahoma) e Ernest Davis (NYU) revisaram o manuscrito e fizeram sugestões úteis. Agradecemos a Ernie Davis, em especial por sua capacidade incansável para ler múltiplos rascunhos e ajudar a melhorar o livro. Nick Hay formatou a bibliografia e no prazo final permaneceu até às 05h30 codificando para melhorar o livro. Jon Barron formatou e melhorou os diagramas nesta edição, enquanto Tim Huang, Mark Paskin e Cynthia Bruyns ajudaram com diagramas e algoritmos em edições anteriores. Ravi Mohan e Ciaran O'Reilly escreveram e mantiveram os exemplos de código Java no site. John Canny escreveu o capítulo de robótica para a primeira edição e Douglas Edwards pesquisou as notas históricas. Tracy Dunkelberger, Allison Michael, Scott Disanno e Jane Bonnell da Pearson fizeram o melhor possível para nos manter dentro do cronograma e deram muitas sugestões úteis. Mais útil de todas foi Julie Sussman, P.P.A., que leu todos os capítulos, proporcionando melhorias extensivas. Nas edições anteriores tivemos revisores que nos avisavam se deixávamos uma vírgula de fora, corrigiam para *qual* quando colocávamos *que*, Julie avisava-nos quando nos esquecíamos de um sinal de menos e corrigia para x_i quando colocávamos x_j. Para cada erro de digitação ou explicação confusa que permaneceu no livro, tenha certeza de que Julie corrigiu pelo menos cinco. Ela perseverou mesmo quando uma falha de energia a obrigou a trabalhar com luz da lanterna, em vez da incandescência do LCD.

Stuart gostaria de agradecer a seus pais pelo apoio e incentivo constante, e à sua esposa, Loy Sheflott, por sua paciência infinita e sabedoria ilimitada. Ele espera que Gordon, Lucy, George e Isaac logo estejam lendo este livro, após perdoá-lo por ter trabalhado tanto. O RUGS (Russell's Unusual Group of Students — Grupo Incomum de Alunos de Russell) foi de uma utilidade sem igual, como sempre.

Peter gostaria de agradecer a seus pais (Torsten e Gerda), os responsáveis pelo início de sua carreira, e também à sua esposa (Kris), a seus filhos (Bella e Juliet), colegas e amigos pelo incentivo e pela tolerância durante as longas horas de escrita e durante as horas ainda mais longas em que foi preciso reescrever algumas páginas.

Nós dois agradecemos aos bibliotecários em Berkeley, Stanford, e à Nasa e aos desenvolvedores do CiteSeer, Wikipédia e Google, que revolucionaram a maneira de pesquisar. Não podemos agradecer a todas as pessoas que utilizaram o livro e fizeram sugestões, mas gostaríamos de observar os comentários especialmente úteis de Gagan Aggarwal, Eyal Amir, Ion Androutsopoulos, Krzysztof Apt, Warren Haley Armstrong, Ellery Aziel, Jeff Van Baalen, Darius Bacon, Brian Baker, Shumeet Baluja, Don Barker, Tony Barrett, James Newton Bass, Don Beal, Howard Beck, Wolfgang Bibel, John Binder, Larry Bookman, David R. Boxall, Ronen Brafman, John Bresina, Gerhard Brewka, Selmer Bringsjord, Carla Brodley, Chris Brown, Emma Brunskill, Wilhelm Burger, Lauren Burka, Carlos Bustamante, João Cachopo, Murray Campbell, Norman Carver, Emmanuel Castro, Anil Chakravarthy, Dan Chisarick, Berthe Choueiry, Roberto Cipolla, David Cohen, James Coleman, Julie Ann Comparini, Corinna Cortes, Gary Cottrell, Ernest Davis, Tom Dean, Rina Dechter, Tom Dietterich, Peter Drake, Chuck Dyer, Doug Edwards, Robert Egginton, Asma'a El-Budrawy, Barbara Engelhardt, Kutluhan Erol, Oren Etzioni, Hana Filip, Douglas Fisher, Jeffrey Forbes, Ken Ford, Eric Fosler-Lussier, John Fosler, Jeremy Frank, Alex Franz, Bob Futrelle, Marek Galecki, Stefan Gerberding, Stuart Gill, Sabine Glesner, Seth Golub, Gosta Grahne, Russ Greiner, Eric Grimson, Barbara Grosz, Larry Hall, Steve Hanks, Othar Hansson, Ernst Heinz, Jim Hendler, Christoph

Herrmann, Paul Hilfinger, Robert Holte, Vasant Honavar, Tim Huang, Seth Hutchinson, Joost Jacob, Mark Jelasity, Magnus Johansson, Istvan Jonyer, Dan Jurafsky, Leslie Kaelbling, Keiji Kanazawa, Surekha Kasibhatla, Simon Kasif, Henry Kautz, Gernot Kerschbaumer, Max Khesin, Richard Kirby, Dan Klein, Kevin Knight, Roland Koenig, Sven Koenig, Daphne Koller, Rich Korf, Benjamin Kuipers, James Kurien, John Lafferty, John Laird, Gus Larsson, John Lazzaro, Jon LeBlanc, Jason Leatherman, Frank Lee, Jon Lehto, Edward Lim, Phil Long, Pierre Louveaux, Don Loveland, Sridhar Mahadevan, Tony Mancill, Jim Martin, Andy Mayer, John McCarthy, David McGrane, Jay Mendelsohn, Risto Miikkulanien, Brian Milch, Steve Minton, Vibhu Mittal, Mehryar Mohri, Leora Morgenstern, Stephen Muggleton, Kevin Murphy, Ron Musick, Sung Myaeng, Eric Nadeau, Lee Naish, Pandu Nayak, Bernhard Nebel, Stuart Nelson, XuanLong Nguyen, Nils Nilsson, Illah Nourbakhsh, Ali Nouri, Arthur Nunes-Harwitt, Steve Omohundro, David Page, David Palmer, David Parkes, Ron Parr, Mark Paskin, Tony Passera, Amit Patel, Michael Pazzani, Fernando Pereira, Joseph Perla, Wim Pijls, Ira Pohl, Martha Pollack, David Poole, Bruce Porter, Malcolm Pradhan, Bill Pringle, Lorraine Prior, Greg Provan, William Rapaport, Deepak Ravichandran, Ioannis Refanidis, Philip Resnik, Francesca Rossi, Sam Roweis, Richard Russell, Jonathan Schaeffer, Richard Scherl, Heinrich Schuetze, Lars Schuster, Bart Selman, Soheil Shams, Stuart Shapiro, Jude Shavlik, Yoram Singer, Satinder Singh, Daniel Sleator, David Smith, Bryan So, Robert Sproull, Lynn Stein, Larry Stephens, Andreas Stolcke, Paul Stradling, Devika Subramanian, Marek Suchenek, Rich Sutton, Jonathan Tash, Austin Tate, Bas Terwijn, Olivier Teytaud, Michael Thielscher, William Thompson, Sebastian Thrun, Eric Tiedemann, Mark Torrance, Randall Upham, Paul Utgoff, Peter van Beek, Hal Varian, Paulina Varshavskaya, Sunil Vemuri, Vandi Verma, Ubbo Visser, Jim Waldo, Toby Walsh, Bonnie Webber, Dan Weld, Michael Wellman, Kamin Whitehouse, Michael Dean White, Brian Williams, David Wolfe, Jason Wolfe, Bill Woods, Alden Wright, Jay Yagnik, Mark Yasuda, Richard Yen, Eliezer Yudkowsky, Weixiong Zhang, Ming Zhao, Shlomo Zilberstein e nossos estimados colegas os Revisores Anônimos.

---

* Conteúdo em inglês.

Sobre os Autores

Stuart Russell nasceu em 1962 em Portsmouth, Inglaterra. Bacharelou-se com louvor em Física pela Universidade de Oxford em 1982, e doutorou-se em Ciência da Computação por Stanford em 1986. Entrou para o corpo docente da Universidade da Califórnia, em Berkeley, onde leciona Ciência da Computação, dirige o Centro para Sistemas Inteligentes e ocupa a cátedra Smith-Zadeh de Engenharia. Em1990, recebeu o Prêmio Presidencial ao Jovem Cientista (*Presidential Young InvestigatorAward*), concedido pela National Science Foundation, e, em 1995, foi covencedor do Prêmio de Computação e Pensamento (*Computers and Thought Award*).

Foi professor da cadeira Miller em 1996 na Universidade de Califórnia, e indicado para a bolsa docente *Chancellor*. Em 1998, foi o palestrante da Conferência Forsythe Memorial, na Universidade de Stanford. É membro efetivo da Associação Americana de Inteligência Artificial e ex-integrante do Conselho Executivo da entidade. Já publicou mais de cem artigos sobre uma ampla gama de tópicos ligados à inteligência artificial. Entre seus outros livros, incluem-se: *The Use of Knowledge in Analogy and Induction* e (com Eric Wefald) *Do the Right Thing: Studies in Limited Rationality*.

Peter Norvig atualmente é diretor de Pesquisa na Google, Inc. e foi o diretor responsável pelos algoritmos de busca do núcleo da Web de 2002 até 2005.

É membro efetivo da Associação Americana de Inteligência Artificial e da Associação para Máquinas de Computação. Anteriormente, foi chefe da Divisão de Ciências Computacionais no Ames Research Center, da NASA, onde supervisionou a pesquisa e o desenvolvimento da robótica e da inteligência artificial para a agência espacial americana. Antes disso, foi cientista-chefe da Junglee, onde ajudou a desenvolver um dos primeiros serviços de acesso a informações pela Internet. Bacharelou-se em matemática aplicada pela Brown University e doutorou-se em Ciência da Computação pela Universidade de Califórnia, em Berkeley. Ele recebeu os prêmios Distinguished Alumni e Engineering Innovation de Berkeley e a Exceptional Achievement Medal da NASA. Tem atuado como professor da Universidade do Sul da Califórnia e pesquisador em Berkeley. Seus outros livros são: *Paradigms of AI Programming: Case Studies in Common Lisp, Verbmobil: A Translation System for Face-to-Face Dialog*, e *Intelligent Help Systems for UNIX*.

Nota dos revisores técnicos da edição brasileira

Participar do projeto editorial de um livro importante como *Inteligência artificial*, de Russel e Norvig, é para nós motivo de orgulho e satisfação, considerando o impacto e a importância dessa obra no ensino de inteligência artificial.

A revisão técnica da tradução é atividade complexa e de grande responsabilidade, pois exige a escolha de termos para a tradução de conceitos técnicos, de forma a facilitar seu entendimento e, ao mesmo tempo, preservar sua vinculação com a forma original, simplificando com isso o acesso do leitor às fontes de conhecimento que permitam prosseguimento de seus estudos.

Para garantir a qualidade da revisão técnica, cada um dos revisores se concentrou nos capítulos relacionados às suas atividades de pesquisa. Assim, a professora Liliane Nunes de Barros revisou os capítulos 2 a 6, 10, 11, 17 e 21; a Professora Renata Wassermann revisou os capítulos 7 a 9, 12, 26 e 27; e o Professor Flávio Soares Corrêa da Silva revisou os capítulos 13 a 16, 18 a 20, 22 a 25 e os Apêndices.

Agradecemos à Elsevier — especialmente à Luciana Félix Macedo e Brunna Prado — pela oportunidade de participar deste projeto, e à Regina Célia Simille de Macedo pela excelente tradução.

Flávio Soares Corrêa da Silva – Professor associado

Liliane Nunes de Barros – Professora associada

Renata Wassermann – Professora associada

Departamento de Ciências da Computação – IME – USP

Sumário

Resolução de problemas

PARTE IV
Conhecimento incerto e pensamento

PARTE I

Inteligência artificial

CAPÍTULO 1

Introdução

Em que tentamos explicar por que consideramos a inteligência artificial um assunto digno de estudo e em que procuramos definir exatamente o que é a inteligência artificial, pois essa definição é importante antes de iniciarmos nosso estudo.

Denominamos nossa espécie *Homo sapiens* — homem sábio — porque nossa **inteligência** é tão importante para nós. Durante milhares de anos, procuramos entender *como pensamos*, isto é, como um mero punhado de matéria pode perceber, compreender, prever e manipular um mundo muito maior e mais complicado que ela própria. O campo da **inteligência artificial**, ou IA, vai ainda mais além: ele tenta não apenas compreender, mas também *construir* entidades inteligentes.

A IA é um dos campos mais recentes em ciências e engenharia. O trabalho começou logo após a Segunda Guerra Mundial, e o próprio nome foi cunhado em 1956. Juntamente com a biologia molecular, a IA é citada regularmente como “o campo em que eu mais gostaria de estar” por cientistas de outras disciplinas. Um aluno de física pode argumentar, com boa dose de razão, que todas as boas ideias já foram desenvolvidas por Galileu, Newton, Einstein e o resto. IA, por outro lado, ainda tem espaço para vários Einsteins e Edisons em tempo integral.

Atualmente, a IA abrange uma enorme variedade de subcampos, do geral (aprendizagem e percepção) até tarefas específicas, como jogos de xadrez, demonstração de teoremas matemáticos, criação de poesia, direção de um carro em estrada movimentada e diagnóstico de doenças. A IA é relevante para qualquer tarefa intelectual; é verdadeiramente um campo universal.

1.1 O QUE É IA?

Afirmamos que a IA é interessante, mas não dissemos o que ela *é*. Na Figura 1.1 podemos visualizar oito definições de IA, dispostas ao longo de duas dimensões. Em linhas gerais, as que estão na parte superior da tabela se relacionam a *processos de pensamento* e *raciocínio*, enquanto as definições da parte inferior se referem ao *comportamento*. As definições do lado esquerdo medem o sucesso em termos de fidelidade ao desempenho *humano*, enquanto as definições do lado direito medem o sucesso comparando-o a um conceito *ideal* de inteligência, chamado de **racionalidade**. Um sistema é racional se “faz a coisa certa”, dado o que ele sabe.

Historicamente, todas as quatro estratégias para o estudo da IA têm sido seguidas, cada uma delas por pessoas diferentes com métodos diferentes. Uma abordagem centrada nos seres humanos deve ser em parte uma ciência empírica, envolvendo hipóteses e confirmação experimental. Uma abordagem racionalista[1] envolve uma combinação de matemática e engenharia. Cada grupo tem ao mesmo tempo desacreditado e ajudado o outro. Vamos examinar as quatro abordagens com mais detalhes.

Pensando como um humano	**Pensando racionalmente**
"O novo e interessante esforço para fazer os computadores pensarem (...) *máquinas com mentes*, no sentido total e literal." (Haugeland, 1985) "[Automatização de] atividades que associamos ao pensamento humano, atividades como a tomada de decisões, a resolução de problemas, o aprendizado..." (Bellman, 1978)	"O estudo das faculdades mentais pelo uso de modelos computacionais." (Charniak e McDermott, 1985) "O estudo das computações que tornam possível perceber, raciocinar e agir." (Winston, 1992)
Agindo como seres humanos	**Agindo racionalmente**
"A arte de criar máquinas que executam funções que exigem inteligência quando executadas por pessoas." (Kurzweil, 1990) "O estudo de como os computadores podem fazer tarefas que hoje são melhor desempenhadas pelas pessoas." (Rich and Knight, 1991)	"Inteligência Computacional é o estudo do projeto de agentes inteligentes." (Poole *et al*., 1998) "AI... está relacionada a um desempenho inteligente de artefatos." (Nilsson, 1998)

Figura 1.1 Algumas definições de inteligência artificial, organizadas em quatro categorias.

1.1.1 Agindo de forma humana: a abordagem do teste de Turing

O **teste de Turing**, proposto por Alan Turing (1950), foi projetado para fornecer uma definição operacional satisfatória de inteligência. O computador passará no teste se um interrogador humano, depois de propor algumas perguntas por escrito, não conseguir descobrir se as respostas escritas vêm de uma pessoa ou de um computador. O Capítulo 26 discute os detalhes do teste e também se um computador seria de fato inteligente se passasse nele. Por enquanto, observamos que programar um computador para passar no teste exige muito trabalho. O computador precisaria ter as seguintes capacidades:

- **processamento de linguagem natural** para permitir que ele se comunique com sucesso em um idioma natural;
- **representação de conhecimento** para armazenar o que sabe ou ouve;
- **raciocínio automatizado** para usar as informações armazenadas com a finalidade de responder a perguntas e tirar novas conclusões;
- **aprendizado de máquina** para se adaptar a novas circunstâncias e para detectar e extrapolar

padrões.

O teste de Turing evitou deliberadamente a interação física direta entre o interrogador e o computador porque a simulação *física* de uma pessoa é desnecessária para a inteligência. Entretanto, o chamado **teste de Turing total** inclui um sinal de vídeo, de forma que o interrogador possa testar as habilidades de percepção do indivíduo, além de oferecer ao interrogador a oportunidade de repassar objetos físicos "pela janelinha". Para ser aprovado no teste de Turing total, o computador precisará de:

- **visão computacional** para perceber objetos e
- **robótica** para manipular objetos e movimentar-se.

Essas seis disciplinas compõem a maior parte da IA, e Turing merece crédito por projetar um teste que permanece relevante depois de 60 anos. Ainda assim, os pesquisadores da IA têm dedicado pouco esforço à aprovação no teste de Turing, acreditando que seja mais importante estudar os princípios básicos da inteligência do que reproduzir um exemplar. O desafio do "voo artificial" teve sucesso quando os irmãos Wright e outros pesquisadores pararam de imitar os pássaros e começaram a usar túneis de vento e aprender sobre aerodinâmia. Os textos de engenharia aeronáutica não definem como objetivo de seu campo criar "máquinas que voem exatamente como pombos a ponto de poderem enganar até mesmo outros pombos".

1.1.2 Pensando de forma humana: a estratégia de modelagem cognitiva

Se pretendemos dizer que dado programa pensa como um ser humano, temos de ter alguma forma de determinar como os seres humanos pensam. Precisamos *penetrar* nos componentes reais da mente humana. Existem três maneiras de fazer isso: através da introspecção — procurando captar nossos próprios pensamentos à medida que eles se desenvolvem — através de experimentos psicológicos — observando uma pessoa em ação; e através de imagens cerebrais, observando o cérebro em ação. Depois que tivermos uma teoria da mente suficientemente precisa, será possível expressar a teoria como um programa de computador. Se os comportamentos de entrada/saída e sincronização do programa coincidirem com os comportamentos humanos correspondentes, isso será a evidência de que alguns dos mecanismos do programa também podem estar operando nos seres humanos. Por exemplo, Allen Newell e Herbert Simon, que desenvolveram o GPS, o "Resolvedor Geral de Problemas" (do inglês "General Problem Solver") (Newell e Simon, 1961), não se contentaram em fazer seu programa resolver problemas de modo correto. Eles estavam mais preocupados em comparar os passos de suas etapas de raciocínio aos passos de indivíduos humanos resolvendo os mesmos problemas. O campo interdisciplinar da **ciência cognitiva** reúne modelos computacionais da IA e técnicas experimentais da psicologia para tentar construir teorias precisas e verificáveis a respeito dos processos de funcionamento da mente humana.

A ciência cognitiva é um campo fascinante por si só, merecedora de diversos livros e de pelo menos uma enciclopédia (Wilson e Keil, 1999). Ocasionalmente, apresentaremos comentários a respeito de semelhanças ou diferenças entre técnicas de IA e a cognição humana. Porém, a ciência

cognitiva de verdade se baseia necessariamente na investigação experimental de seres humanos ou animais. Deixaremos esse assunto para outros livros à medida que supomos que o leitor tenha acesso somente a um computador para realizar experimentação.

Nos primórdios da IA, frequentemente havia confusão entre as abordagens: um autor argumentava que um algoritmo funcionava bem em uma tarefa e que, *portanto*, era um bom modelo de desempenho humano ou vice-versa. Os autores modernos separam os dois tipos de afirmações; essa distinção permitiu que tanto a IA quanto a ciência cognitiva se desenvolvessem com maior rapidez. Os dois campos continuam a fertilizar um ao outro, principalmente na visão computacional, que incorpora evidências neurofisiológicas em modelos computacionais.

1.1.3 Pensando racionalmente: a abordagem das "leis do pensamento"

O filósofo grego Aristóteles foi um dos primeiros a tentar codificar o "pensamento correto", isto é, os processos de raciocínio irrefutáveis. Seus **silogismos** forneceram padrões para estruturas de argumentos que sempre resultavam em conclusões corretas ao receberem premissas corretas — por exemplo, "Sócrates é um homem; todos os homens são mortais; então, Sócrates é mortal". Essas leis do pensamento deveriam governar a operação da mente; seu estudo deu início ao campo chamado **lógica**.

Os lógicos do século XIX desenvolveram uma notação precisa para declarações sobre todos os tipos de coisas no mundo e sobre as relações entre elas (compare isso com a notação aritmética básica, que fornece apenas declarações a respeito de números). Por volta de 1965, existiam programas que, em princípio, podiam resolver *qualquer* problema solucionável descrito em notação lógica (contudo, se não houver solução, o programa poderá entrar num laço infinito). A chamada tradição **logicista** dentro da inteligência artificial espera desenvolver tais programas para criar sistemas inteligentes.

Essa abordagem enfrenta dois obstáculos principais. Primeiro, não é fácil enunciar o conhecimento informal nos termos formais exigidos pela notação lógica, em particular quando o conhecimento é menos de 100% certo. Em segundo lugar, há uma grande diferença entre ser capaz de resolver um problema "em princípio" e resolvê-lo na prática. Até mesmo problemas com apenas algumas centenas de fatos podem esgotar os recursos computacionais de qualquer computador, a menos que ele tenha alguma orientação sobre as etapas de raciocínio que deve tentar primeiro. Embora ambos os obstáculos se apliquem a *qualquer* tentativa de construir sistemas de raciocínio computacional, eles surgiram primeiro na tradição logicista.

1.1.4 Agindo racionalmente: a abordagem de agente racional

Um **agente** é simplesmente algo que age (a palavra *agente* vem do latino *agere*, que significa fazer). Certamente todos os programas de computador realizam alguma coisa, mas espera-se que um agente computacional faça mais: opere sob controle autônomo, perceba seu ambiente, persista por um período de tempo prolongado, adapte-se a mudanças e seja capaz de criar e perseguir metas. Um

agente racional é aquele que age para alcançar o melhor resultado ou, quando há incerteza, o melhor resultado esperado.

Na abordagem de “leis do pensamento” para IA, foi dada ênfase a inferências corretas. Às vezes, a realização de inferências corretas é uma *parte* daquilo que caracteriza um agente racional porque uma das formas de agir racionalmente é raciocinar de modo lógico até a conclusão de que dada ação alcançará as metas pretendidas e, depois, agir de acordo com essa conclusão. Por outro lado, a inferência correta não representa *toda* a racionalidade; em algumas situações, não existe nenhuma ação comprovadamente correta a realizar, mas mesmo assim algo tem de ser feito. Também existem modos de agir racionalmente que não se pode dizer que envolvem inferências. Por exemplo, afastar-se de um fogão quente é um ato reflexo, em geral mais bem-sucedido que uma ação mais lenta executada após cuidadosa deliberação.

Todas as habilidades necessárias à realização do teste de Turing também permitem que o agente haja racionalmente. Representação do conhecimento e raciocínio permitem que os agentes alcancem boas decisões. Precisamos ter a capacidade de gerar sentenças compreensíveis em linguagem natural porque enunciar essas sentenças nos ajuda a participar de uma sociedade complexa. Precisamos aprender não apenas por erudição, mas também para melhorar nossa habilidade de gerar comportamento efetivo.

A abordagem do agente racional tem duas vantagens sobre as outras abordagens. Primeiro, ela é mais geral que a abordagem de “leis do pensamento” porque a inferência correta é apenas um dentre vários mecanismos possíveis para se alcançar a racionalidade. Em segundo lugar, ela é mais acessível ao desenvolvimento científico do que as estratégias baseadas no comportamento ou no pensamento humano. O padrão de racionalidade é matematicamente bem definido e completamente geral, podendo ser “desempacotado” para gerar modelos de agente que comprovadamente irão atingi-lo. Por outro lado, o comportamento humano está bem adaptado a um ambiente específico e é definido como a soma de tudo o que os humanos fazem. *Portanto, este livro se concentrará nos princípios gerais de agentes racionais e nos componentes para construí-los*. Veremos que, apesar da aparente simplicidade com que o problema pode ser enunciado, surge uma enorme variedade de questões quando tentamos resolvê-lo. O Capítulo 2 descreve algumas dessas questões com mais detalhes.

Devemos ter em mente um ponto importante: logo veremos que alcançar a racionalidade perfeita — sempre fazer a coisa certa — não é algo viável em ambientes complicados. As demandas computacionais são demasiado elevadas. Porém, na maior parte do livro, adotaremos a hipótese de trabalho de que a racionalidade perfeita é um bom ponto de partida para a análise. Ela simplifica o problema e fornece a configuração apropriada para a maioria do material básico na área. Os Capítulos 5 e 17 lidam explicitamente com a questão da **racionalidade limitada** — agir de forma apropriada quando não existe tempo suficiente para realizar todas as computações que gostaríamos de fazer.

1.2 OS FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL

Nesta seção, apresentaremos um breve histórico das disciplinas que contribuíram com ideias,

pontos de vista e técnicas para a IA. Como qualquer histórico, este foi obrigado a se concentrar em um pequeno número de pessoas, eventos e ideias, e ignorar outros que também eram importantes. Organizamos o histórico em torno de uma série de perguntas. Certamente, não desejaríamos dar a impressão de que essas questões são as únicas de que as disciplinas tratam ou que todas as disciplinas estejam se encaminhando para a IA como sua realização final.

1.2.1 Filosofia

- Regras formais podem ser usadas para obter conclusões válidas?
- Como a mente (o intelecto) se desenvolve a partir de um cérebro físico?
- De onde vem o conhecimento?
- Como o conhecimento conduz à ação?

Aristóteles (384-322 a.C.), cujo busto aparece na capa deste livro, foi o primeiro a formular um conjunto preciso de leis que governam a parte racional da mente. Ele desenvolveu um sistema informal de silogismos para raciocínio apropriado que, em princípio, permitiam gerar conclusões mecanicamente, dadas as premissas iniciais. Muito mais tarde, Ramon Lull (1315) apresentou a ideia de que o raciocínio útil poderia na realidade ser conduzido por um artefato mecânico. Thomas Hobbes (1588-1679) propôs que o raciocínio era semelhante à computação numérica, ou seja, que "efetuamos somas e subtrações em nossos pensamentos silenciosos". A automação da própria computação já estava bem próxima; por volta de 1500, Leonardo da Vinci (1452-1519) projetou, mas não construiu, uma calculadora mecânica; reconstruções recentes mostraram que o projeto era funcional. A primeira máquina de calcular conhecida foi construída em torno de 1623 pelo cientista alemão Wilhelm Schickard (1592-1635), embora a Pascaline, construída em 1642 por Blaise Pascal (1623-1662), seja mais famosa. Pascal escreveu que "a máquina aritmética produz efeitos que parecem mais próximos ao pensamento que todas as ações dos animais". Gottfried Wilhelm Leibnitz (1646-1716) construiu um dispositivo mecânico destinado a efetuar operações sobre conceitos, e não sobre números, mas seu escopo era bastante limitado. Leibnitz superou Pascal através da construção de uma calculadora que podia somar, subtrair, multiplicar e extrair raízes, enquanto a Pascaline só podia adicionar e subtrair. Alguns especularam que as máquinas não poderiam fazer apenas cálculos, mas realmente ser capazes de pensar e agir por conta própria. Em seu livro de 1651, *Leviatã*, Thomas Hobbes sugeriu a ideia de um "animal artificial", argumentando: "Pois o que é o coração, senão uma mola; e os nervos, senão tantas cordas; e as articulações, senão tantas rodas."

Dizer que a mente opera, pelo menos em parte, de acordo com regras lógicas e construir sistemas físicos que emulam algumas dessas regras é uma coisa; outra é dizer que a mente em si *é* esse sistema físico. René Descartes (1596-1650) apresentou a primeira discussão clara da distinção entre mente e matéria, e dos problemas que surgem dessa distinção. Um dos problemas relacionados com uma concepção puramente física da mente é o fato de que ela parece deixar pouco espaço para o livre-arbítrio: se a mente é governada inteiramente por leis físicas, então ela não tem mais livre-arbítrio que uma pedra que "decide" cair em direção ao centro da Terra. Descartes advogava fortemente a favor do poder da razão em entender o mundo, uma filosofia hoje chamada de **racionalismo**, e que tinha Aristóteles e Leibnitz como membros. Descartes também era um proponente do **dualismo**. Ele

sustentava que havia uma parte da mente humana (ou alma, ou espírito) que transcende a natureza, isenta das leis físicas. Por outro lado, os animais não possuem essa qualidade dual; eles podiam ser tratados como máquinas. Uma alternativa para o dualismo é o **materialismo**. O materialismo sustenta que a operação do cérebro de acordo com as leis da física *constitui* a mente. O livre-arbítrio é simplesmente o modo como a percepção das escolhas disponíveis se mostra para a entidade que escolhe.

Dada uma mente física que manipula o conhecimento, o próximo problema é estabelecer a origem do conhecimento. O movimento chamado **empirismo**, iniciado a partir da obra de Francis Bacon (1561-1626), *Novum Organum*,[2] se caracterizou por uma frase de John Locke (1632-1704): "Não há nada na compreensão que não estivesse primeiro nos sentidos." A obra de David Hume (1711-1776), *A Treatise of Human Nature* (Hume, 1739) propôs aquilo que se conhece hoje como o princípio de **indução**: as regras gerais são adquiridas pela exposição a associações repetidas entre seus elementos. Com base no trabalho de Ludwig Wittgenstein (1889-1951) e Bertrand Russell (1872-1970), o famoso Círculo de Viena, liderado por Rudolf Carnap (1891-1970), desenvolveu a doutrina do **positivismo lógico**. Essa doutrina sustenta que todo conhecimento pode ser caracterizado por teorias lógicas conectadas, em última análise, a **sentenças de observação** que correspondem a entradas sensoriais; desse modo, o positivismo lógico combina o racionalismo e o empirismo.[3] A **teoria da confirmação** de Carnap e Carl Hempel (1905-1997) tentava compreender a aquisição do conhecimento através da experiência. O livro de Carnap, *The Logical Structure of the World* (1928), definiu um procedimento computacional explícito para extrair conhecimento de experiências elementares. Provavelmente, foi a primeira teoria da mente como um processo computacional.

O último elemento no quadro filosófico da mente é a conexão entre conhecimento e ação. Essa questão é vital para a IA porque a inteligência exige ação, bem como raciocínio. Além disso, apenas pela compreensão de como as ações são justificadas podemos compreender como construir um agente cujas ações sejam justificáveis (ou racionais). Aristóteles argumentava (no *De Motu Animalium*) que as ações se justificam por uma conexão lógica entre metas e conhecimento do resultado da ação (a última parte deste extrato também aparece na capa deste livro, no original em grego):

> Porém, como explicar que o pensamento às vezes esteja acompanhado pela ação e às vezes não, às vezes esteja acompanhado pelo movimento e outras vezes não? Aparentemente, acontece quase o mesmo no caso do raciocínio e na realização de inferências sobre objetos imutáveis. Contudo, nesse caso o fim é uma proposição especulativa (…) enquanto aqui a conclusão que resulta das duas premissas é uma ação. (…) Preciso me cobrir; um casaco é uma coberta. Preciso de um casaco. O que eu preciso, tenho de fazer; preciso de um casaco. Tenho de fazer um casaco. E a conclusão, "tenho de fazer um casaco", é uma ação.

Na obra *Ética a Nicômaco* (Livro III. 3, 1112b), Aristóteles desenvolve esse tópico um pouco mais, sugerindo um algoritmo:

> Não deliberamos sobre os fins, mas sobre os meios. Um médico não delibera sobre se deve ou não curar nem um orador sobre se deve ou não persuadir, (…) Eles dão a finalidade por estabelecida e procuram saber a maneira de alcançá-la; se lhes parece poder ser alcançada por vários meios,

procuram saber o mais fácil e o mais eficaz; e se há apenas um meio para alcançá-la, procuram saber *como* será alcançada por esse meio e por que outro meio alcançar *esse* primeiro, até chegar ao primeiro princípio, que é o último na ordem de descoberta. (…) e o que vem em último lugar na ordem da análise parece ser o primeiro na ordem da execução. E, se chegarmos a uma impossibilidade, abandonamos a busca; por exemplo, se precisarmos de dinheiro e não for possível consegui-lo; porém, se algo parecer possível, tentaremos realizá-lo.[4]

O algoritmo de Aristóteles foi implementado 2.300 anos mais tarde, por Newell e Simon, em seu programa GPS. Agora, poderíamos denominá-lo sistema de planejamento regressivo (ver o Capítulo 10.)

A análise baseada em metas é útil, mas não nos diz o que fazer quando várias ações alcançarem a meta ou quando nenhuma ação a alcançar por completo. Antoine Arnauld (1612-1694) descreveu corretamente uma fórmula quantitativa para definir que ação executar em casos como esse (ver o Capítulo 16). O livro de John Stuart Mill (1806-1873), *Utilitarianism* (Mill, 1863), promoveu a ideia de critérios de decisão racionais em todas as esferas da atividade humana. A teoria de decisões é mais formalmente discutida na próxima seção.

1.2.2 Matemática

- Quais são as regras formais para obter conclusões válidas?
- O que pode ser computado?
- Como raciocinamos com informações incertas?

Os filósofos demarcaram a maioria das ideias importantes sobre a IA, mas o salto para uma ciência formal exigiu certo nível de formalização matemática em três áreas fundamentais: lógica, computação e probabilidade.

A ideia de lógica formal pode ser traçada até os filósofos da Grécia antiga, mas seu desenvolvimento matemático começou realmente com o trabalho de George Boole (1815-1864), que definiu os detalhes da lógica proposicional ou lógica booleana (Boole, 1847). Em 1879, Gottlob Frege (1848-1925) estendeu a lógica de Boole para incluir objetos e relações, criando a lógica de primeira ordem que é utilizada hoje.[5] Alfred Tarski (1902-1983) introduziu uma teoria de referência que mostra como relacionar os objetos de uma lógica a objetos do mundo real.

A próxima etapa foi determinar os limites do que poderia ser feito com a lógica e a computação. Acredita-se que o primeiro **algoritmo** não trivial seja o algoritmo de Euclides para calcular o maior divisor comum. A palavra *algoritmo* (e a ideia de estudá-lo) vem de Al-Khowarazmi, um matemático persa do século IX, cujos escritos também introduziram os numerais arábicos e a álgebra na Europa. Boole e outros discutiram algoritmos para dedução lógica e, no final do século XIX, foram empreendidos esforços para formalizar o raciocínio matemático geral como dedução lógica. Em 1930, Kurt Gödel (1906-1978) mostrou que existe um procedimento efetivo para provar qualquer afirmação verdadeira na lógica de primeira ordem de Frege e Russell, mas essa lógica não poderia captar o princípio de indução matemática necessário para caracterizar os números naturais. Em 1931, Gödel mostrou que existem de fato limites sobre dedução. Seu **teorema da incompletude** mostrou

que, em qualquer teoria formal tão forte como a aritmética de Peano (a teoria elementar dos números naturais), existem afirmações verdadeiras que são indecidíveis no sentido de que não existem provas na teoria.

Esse resultado fundamental também pode ser interpretado como a demonstração de que existem algumas funções sobre os inteiros que não podem ser representadas por um algoritmo, isto é, não podem ser calculadas. Isso motivou Alan Turing (1912-1954) a tentar caracterizar exatamente que funções *são* **computáveis** — capazes de ser computáveis. Na realidade, essa noção é ligeiramente problemática porque a noção de computação ou de procedimento efetivo realmente não pode ter uma definição formal. No entanto, a tese de Church-Turing, que afirma que a máquina de Turing (Turing, 1936) é capaz de calcular qualquer função computável, em geral é aceita como definição suficiente. Turing também mostrou que existiam algumas funções que nenhuma máquina de Turing poderia calcular. Por exemplo, nenhuma máquina pode determinar, *de forma geral*, se dado programa retornará uma resposta sobre certa entrada ou se continuará funcionando para sempre.

Embora a decidibilidade e a computabilidade sejam importantes para a compreensão da computação, a noção de **tratabilidade** teve um impacto muito maior. Em termos gerais, um problema é chamado de intratável se o tempo necessário para resolver instâncias dele cresce exponencialmente com o tamanho das instâncias. A distinção entre crescimento polinomial e exponencial da complexidade foi enfatizada primeiro em meados da década de 1960 (Cobham, 1964; Edmonds, 1965). Ela é importante porque o crescimento exponencial significa que até mesmo instâncias moderadamente grandes não podem ser resolvidas em qualquer tempo razoável. Portanto, devemos procurar dividir o problema global de geração de comportamento inteligente em subproblemas tratáveis, em vez de subproblemas intratáveis.

Como é possível reconhecer um problema intratável? A teoria da **NP-completude**, apresentada primeiro por Steven Cook (1971) e Richard Karp (1972), fornece um método. Cook e Karp demonstraram a existência de grandes classes de problemas canônicos de busca combinatória e de raciocínio que são NP-completos. Qualquer classe de problemas à qual a classe de problemas NP-completos pode ser reduzida provavelmente é intratável (embora não tenha sido provado que problemas NP-completos são necessariamente intratáveis, a maioria dos teóricos acredita nisso). Esses resultados contrastam com o otimismo com que a imprensa popular saudou os primeiros computadores — “Supercérebros eletrônicos” que eram “Mais rápidos que Einstein!”. Apesar da crescente velocidade dos computadores, o uso parcimonioso de recursos é que caracterizará os sistemas inteligentes. *Grosso modo*, o mundo é uma instância de um problema *extremamente* grande! Trabalhar com IA ajudou a explicar por que algumas instâncias de problemas NP-completos são difíceis, enquanto outras são fáceis (Cheeseman *et al.*, 1991).

Além da lógica e da computação, a terceira grande contribuição da matemática para a IA é a teoria da **probabilidade**. O italiano Gerolamo Cardano (1501-1576) foi o primeiro a conceber a ideia de probabilidade, descrevendo-a em termos dos resultados possíveis de jogos de azar. Em 1654, Blaise Pascal (1623-1662), numa carta para Pierre Fermat (1601-1665), mostrou como predizer o futuro de um jogo de azar inacabado e atribuir recompensas médias aos jogadores. A probabilidade se transformou rapidamente em uma parte valiosa de todas as ciências quantitativas, ajudando a lidar com medidas incertas e teorias incompletas. James Bernoulli (1654-1705), Pierre Laplace (1749-1827) e outros pesquisadores aperfeiçoaram a teoria e introduziram novos métodos estatísticos.

Thomas Bayes (1702-1761), que aparece na capa deste livro, propôs uma regra para atualizar probabilidades à luz de novas evidências. A regra de Bayes e o campo resultante chamado análise bayesiana formam a base da maioria das abordagens modernas para raciocínio incerto em sistemas de IA.

1.2.3 Economia

- Como devemos tomar decisões para maximizar a recompensa?
- Como devemos fazer isso quando outros não podem nos acompanhar?
- Como devemos fazer isso quando a recompensa pode estar distante no futuro?

A ciência da economia teve início em 1776, quando o filósofo escocês Adam Smith (1723-1790) publicou *An Inquiry into the Nature and Causes of the Wealth of Nations*. Embora os antigos gregos e outros filósofos tenham contribuído para o pensamento econômico, Smith foi o primeiro a tratá-lo como ciência, usando a ideia de que podemos considerar que as economias consistem em agentes individuais que maximizam seu próprio bem-estar econômico. A maioria das pessoas pensa que a economia trata de dinheiro, mas os economistas dirão que, na realidade, a economia estuda como as pessoas fazem escolhas que levam a resultados preferenciais. Quando o McDonalds oferece um hambúrguer por um dólar, está afirmando que prefere o dólar e espera que os clientes prefiram o hambúrguer. O tratamento matemático de “resultados preferenciais” ou **utilidade** foi formalizado primeiro por Léon Walras (1834-1910) e aperfeiçoado por Frank Ramsey (1931) e, mais tarde, por John von Neumann e Oskar Morgenstern em seu livro *The Theory of Games and Economic Behavior* (1944).

A **teoria da decisão**, que combina a teoria da probabilidade com a teoria da utilidade, fornece uma estrutura formal e completa para decisões (econômicas ou outras) tomadas sob a incerteza, ou seja, em casos nos quais as descrições probabilísticas captam de forma apropriada o ambiente do tomador de decisões. Isso é adequado para “grandes” economias em que cada agente não precisa levar em conta as ações de outros agentes como indivíduos. No caso das “pequenas” economias, a situação é muito mais parecida com um **jogo**: as ações de um jogador podem afetar de forma significativa a utilidade de outro (positiva ou negativamente). O desenvolvimento da **teoria dos jogos** por Von Neumann e Morgenstern (consulte também Luce e Raiffa, 1957) incluiu o surpreendente resultado de que, em alguns jogos, um agente racional deve adotar políticas que são (ou pelo menos parecem ser) aleatórias. Ao contrário da teoria da decisão, a teoria dos jogos não oferece uma receita inequívoca para a seleção de ações.

De modo geral, os economistas não trataram a terceira questão da listagem anterior, ou seja, como tomar decisões racionais quando as recompensas das ações não são imediatas, mas resultam de várias ações executadas *em sequência*. Esse tópico foi adotado no campo de **pesquisa operacional**, que emergiu na Segunda Guerra Mundial dos esforços britânicos para otimizar instalações de radar e, mais tarde, encontrou aplicações civis em decisões complexas de administração. O trabalho de Richard Bellman (1957) formalizou uma classe de problemas de decisão sequencial chamados **processos de decisão de Markov**, que estudaremos nos Capítulos 17 e 21.

O trabalho em economia e pesquisa operacional contribuiu muito para nossa noção de agentes racionais, ainda que por muitos anos a pesquisa em IA se desenvolvesse ao longo de caminhos inteiramente separados. Uma razão para isso era a aparente **complexidade** da tomada de decisões racionais. Herbert Simon (1916-2001), o pesquisador pioneiro da IA, ganhou o Prêmio Nobel de economia em 1978 por seu trabalho inicial demonstrando que modelos baseados em **satisfação** — a tomada de decisões "boas o suficiente", em vez de calcular laboriosamente uma decisão ótima — forneciam uma descrição melhor do comportamento humano real (Simon, 1947). Desde os anos 1990, ressurgiu o interesse pelas técnicas da teoria da decisão para sistemas de agentes (Wellman, 1995).

1.2.4 Neurociência

- Como o cérebro processa informações?

A **neurociência** é o estudo do sistema nervoso, em particular do cérebro. Apesar de o modo exato como o cérebro habilita o pensamento ser um dos grandes mistérios da ciência, o fato de ele *habilitar* o pensamento foi avaliado por milhares de anos devido à evidência de que pancadas fortes na cabeça podem levar à incapacitação mental. Também se sabe há muito tempo que o cérebro dos seres humanos tem algumas características diferentes; em aproximadamente 335 a.C., Aristóteles escreveu: "De todos os animais, o homem é o que tem o maior cérebro em proporção ao seu tamanho."[6] Ainda assim, apenas em meados do século XVIII o cérebro foi amplamente reconhecido como a sede da consciência. Antes disso, acreditava-se que a sede da consciência poderia estar localizada no coração e no baço.

O estudo da afasia (deficiência da fala) feito por Paul Broca (1824-1880) em 1861, com pacientes cujo cérebro foi danificado, demonstrou a existência de áreas localizadas do cérebro responsáveis por funções cognitivas específicas. Em particular, ele mostrou que a produção da fala estava localizada em uma parte do hemisfério cerebral esquerdo agora chamada área de Broca.[7] Nessa época, sabia-se que o cérebro consistia em células nervosas ou **neurônios**, mas apenas em 1873 Camillo Golgi (1843-1926) desenvolveu uma técnica de coloração que permitiu a observação de neurônios individuais no cérebro (ver a Figura 1.2). Essa técnica foi usada por Santiago Ramon y Cajal (1852-1934) em seus estudos pioneiros das estruturas de neurônios do cérebro.[8] Nicolas Rashevsky (1936, 1938) foi o primeiro a aplicar modelos matemáticos ao estudo do sistema nervoso.

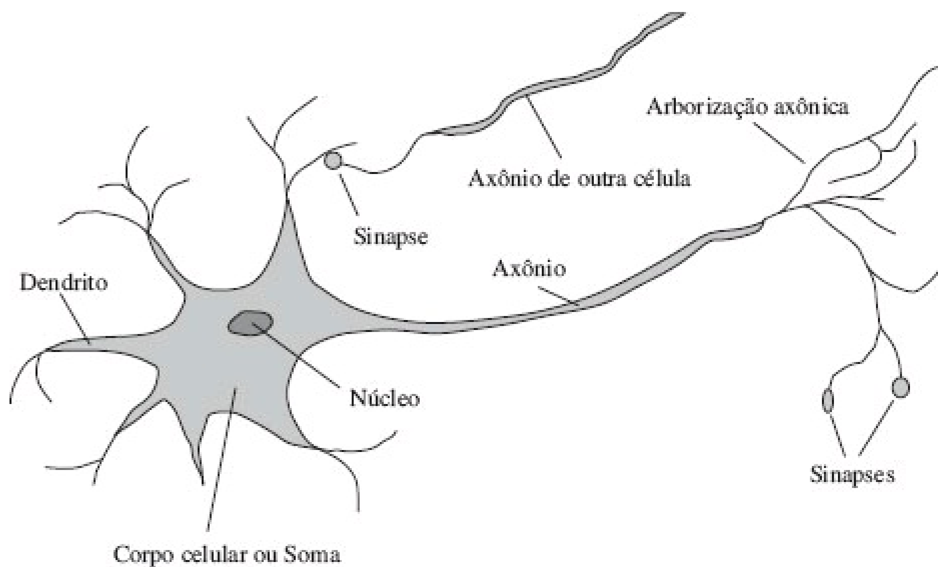


Figura 1.2 Partes de uma célula nervosa ou neurônio. Cada neurônio consiste em um corpo celular ou soma, que contém um núcleo celular. Ramificando-se a partir do corpo celular, há uma série de fibras chamadas dendritos e uma única fibra longa chamada axônio. O axônio se estende por uma longa distância, muito mais longa do que indica a escala desse diagrama. Em geral, um axônio têm 1 cm de comprimento (100 vezes o diâmetro do corpo celular), mas pode alcançar até 1 metro. Um neurônio faz conexões com 10-100.000 outros neurônios, em junções chamadas sinapses. Os sinais se propagam de um neurônio para outro por meio de uma complicada reação eletroquímica. Os sinais controlam a atividade cerebral em curto prazo e também permitem mudanças a longo prazo na posição e na conectividade dos neurônios. Acredita-se que esses mecanismos formem a base para o aprendizado no cérebro. A maior parte do processamento de informações ocorre no córtex cerebral, a camada exterior do cérebro. A unidade organizacional básica parece ser uma coluna de tecido com aproximadamente 0,5 mm de diâmetro, contendo cerca de 20.000 neurônios e estendendo-se por toda a profundidade do córtex, cerca de 4 mm nos seres humanos.

Atualmente, temos alguns dados sobre o mapeamento entre áreas do cérebro e as partes do corpo que elas controlam ou das quais recebem entrada sensorial. Tais mapeamentos podem mudar radicalmente no curso de algumas semanas, e alguns animais parecem ter vários mapas. Além disso, não compreendemos inteiramente como outras áreas do cérebro podem assumir o comando de certas funções quando uma área é danificada. Praticamente não há teoria que explique como a memória de um indivíduo é armazenada.

A medição da atividade do cérebro intacto teve início em1929, com a invenção do eletroencefalógrafo (EEG) por Hans Berger. O desenvolvimento recente do processamento de imagens por ressonância magnética funcional (fMRI — *functional Magnetic Resonance Imaging*) (Ogawa *et al*., 1990; Cabeza e Nyberg, 2001) está dando aos neurocientistas imagens sem precedentes de detalhes da atividade do cérebro, tornando possíveis medições que correspondem em aspectos interessantes a processos cognitivos em ação. Essas medições são ampliadas por avanços na gravação da atividade dos neurônios em uma única célula. Os neurônios individuais podem ser estimulados eletricamente, quimicamente ou mesmo opticamente (Han e Boyden, 2007), permitindo que os relacionamentos neuronais de entrada-saída sejam mapeados. Apesar desses avanços, ainda estamos longe de compreender como realmente funciona qualquer desses processos cognitivos.

 A conclusão verdadeiramente espantosa é que *uma coleção de células simples pode levar ao pensamento*, *à ação e à consciência* ou, nas palavras incisivas de John Searle (1992), *os cérebros geram mentes*. A única teoria alternativa real é o misticismo, que significa operar em algum reino místico que está além da ciência física.

De alguma forma cérebros e computadores digitais têm propriedades diferentes. A Figura 1.3 mostra que os computadores têm um ciclo de tempo que é um milhão de vezes mais rápido que o cérebro. O cérebro é composto por muito mais capacidade de armazenamento e interconexões que um computador pessoal de última geração, apesar de os maiores supercomputadores apresentarem uma capacidade similar a do cérebro. Entretanto, observe que o cérebro não parece usar todos os seus neurônios simultaneamente. Os futuristas enaltecem demais esses números, apontando para uma próxima **singularidade** em que os computadores alcançariam um nível sobrehumano de performance (Vinge, 1993; Kurzweil, 2005), mas as comparações cruas não são especialmente informativas. Mesmo com um computador com capacidade ilimitada, não saberíamos como atingir o nível de inteligência do cérebro.

	Supercomputador	**Computador pessoal**	**Mente humana**
Unidades computacionais	10^4 CPUs, 10^{12} transistores	4 CPUs, 10^9 transistores	10^{11} neurônios
Unidades de armazenamaneto	10^{14} bits RAM 10^{15} bits disco	10^{11} bits 10^{13} RAM bits disco	10^{11} neurônios 10^{14} sinapses
Tempo de ciclo	10^{-9} seg	10^{-9} seg	10^{-3} seg
Operações/seg	10^{15}	10^{10}	10^{17}
Atualizações de memória/seg	10^{14}	10^{10}	10^{14}

Figura 1.3 Comparação grosseira dos recursos computacionais brutos disponíveis entre o supercomputador Blue Gene da IBM, um computador pessoal típico de 2008 e o cérebro humano. Os números do cérebro são fixos essencialmente, enquanto os números do supercomputador crescem por um fator de 10, mais ou menos a cada cinco anos, permitindo-lhe alcançar paridade aproximada com o cérebro. O computador pessoal está atrasado em todas as métricas, exceto no tempo de ciclo.

1.2.5 Psicologia

- Como os seres humanos e os animais pensam e agem?

Normalmente, considera-se que as origens da psicologia científica remontam ao trabalho do físico alemão Hermann von Helmholtz (1821-1894) e de seu aluno Wilhelm Wundt (1832-1920). Helmholtz aplicou o método científico ao estudo da visão humana, e seu *Handbook of Physiological Optics* é descrito até hoje como “o mais importante tratado sobre a física e a fisiologia da visão humana” (Nalwa, 1993, p. 15). Em 1879, Wundt abriu o primeiro laboratório de psicologia experimental na

Universidade de Leipzig. Ele insistia em experimentos cuidadosamente controlados, nos quais seus colaboradores executariam uma tarefa perceptiva ou associativa enquanto refletiam sobre seus processos de pensamento. O controle cuidadoso percorreu um longo caminho para transformar a psicologia em ciência, mas a natureza subjetiva dos dados tornava improvável que um pesquisador divergisse de suas próprias teorias. Por outro lado, os biólogos que estudavam o comportamento animal careciam de dados introspectivos e desenvolveram uma metodologia objetiva, como descreveu H. S. Jennings (1906) em seu influente trabalho *Behavior of the Lower Organisms*. Aplicando esse ponto de vista aos seres humanos, o movimento chamado **behaviorismo**, liderado por John Watson (1878-1958), rejeitava *qualquer* teoria que envolvesse processos mentais com base no fato de que a introspecção não poderia fornecer evidência confiável. Os behavioristas insistiam em estudar apenas medidas objetivas das percepções (ou *estímulos*) dados a um animal e suas ações resultantes (ou *respostas*). O behaviorismo descobriu muito sobre ratos e pombos, mas teve menos sucesso na compreensão dos seres humanos.

A visão do cérebro como um dispositivo de processamento de informações, uma característica importante da **psicologia cognitiva**, tem suas origens nos trabalhos de William James (1842-1910). Helmholtz também insistiu que a percepção envolvia uma forma de inferência lógica inconsciente. O ponto de vista cognitivo foi em grande parte eclipsado pelo behaviorismo nos Estados Unidos, mas na Unidade de Psicologia Aplicada de Cambridge, dirigida por Frederic Bartlett (1886-1969), a modelagem cognitiva foi capaz de florescer. *The Nature of Explanation*, de Kenneth Craik (1943), aluno e sucessor de Bartlett, restabeleceu com vigor a legitimidade de termos "mentais" como crenças e objetivos, argumentando que eles são tão científicos quanto, digamos, usar a pressão e a temperatura ao falar sobre gases, apesar de eles serem constituídos por moléculas que não têm nenhuma dessas duas propriedades. Craik especificou os três passos fundamentais de um agente baseado no conhecimento: (1) o estímulo deve ser traduzido em uma representação interna, (2) a representação é manipulada por processos cognitivos para derivar novas representações internas e (3) por sua vez, essas representações são de novo traduzidas em ações. Ele explicou com clareza por que esse era um bom projeto de um agente:

> Se o organismo transporta um "modelo em escala reduzida" da realidade externa e de suas próprias ações possíveis dentro de sua cabeça, ele é capaz de experimentar várias alternativas, concluir qual a melhor delas, reagir a situações futuras antes que elas surjam, utilizar o conhecimento de eventos passados para lidar com o presente e o futuro e, em todos os sentidos, reagir de maneira muito mais completa, segura e competente às emergências que enfrenta. (Craik, 1943)

Após a morte de Craik, em um acidente de bicicleta em 1945, seu trabalho teve continuidade com Donald Broadbent, cujo livro *Perception and Communication* (1958) foi um dos primeiros trabalhos a modelar fenômenos psicológicos como processamento de informações. Enquanto isso, nos Estados Unidos, o desenvolvimento da modelagem de computadores levou à criação do campo da **ciência cognitiva**. Pode-se dizer que o campo teve início em um seminário em setembro de 1956 no MIT (veremos que esse seminário ocorreu apenas dois meses após a conferência em que a própria IA "nasceu"). No seminário, George Miller apresentou *The Magic Number Seven*, Noam Chomsky apresentou *Three Models of Language* e Allen Newell e Herbert Simon apresentaram *The Logic*

Theory Machine. Esses três documentos influentes mostraram como modelos de computadores podiam ser usados para tratar a psicologia da memória, a linguagem e o pensamento lógico, respectivamente. Agora é comum entre os psicólogos a visão de que “uma teoria cognitiva deve ser como um programa de computador” (Anderson, 1980), isto é, ela deve descrever um mecanismo detalhado de processamento de informações por meio do qual alguma função cognitiva poderia ser implementada.

1.2.6 Engenharia de computadores

- Como podemos construir um computador eficiente?

Para a inteligência artificial ter sucesso, precisamos de inteligência e de um artefato. O computador tem sido o artefato preferido. O computador eletrônico digital moderno foi criado independentemente e quase ao mesmo tempo por cientistas de três países que participavam da Segunda Guerra Mundial. O primeiro computador *operacional* foi a máquina eletromecânica de Heath Robinson,[9] construída em 1940 pela equipe de Alan Turing com um único propósito: decifrar mensagens alemãs. Em 1943, o mesmo grupo desenvolveu o Colossus, uma poderosa máquina de uso geral baseada em válvulas eletrônicas.[10] O primeiro computador *programável* operacional foi o Z-3, criado por Konrad Zuse na Alemanha, em 1941. Zuse também criou os números de ponto flutuante e a primeira linguagem de programação de alto nível, denominada Plankalkul. O primeiro computador *eletrônico*, o ABC, foi montado por John Atanasoff e por seu aluno Clifford Berry, entre 1940 e 1942, na Iowa State University. A pesquisa de Atanasoff recebeu pouco apoio ou reconhecimento; foi o ENIAC, desenvolvido como parte de um projeto militar secreto na University of Pennsylvania por uma equipe que incluía John Mauchly e John Eckert, que provou ser o precursor mais influente dos computadores modernos.

Desde aquele tempo, cada geração de hardware de computador trouxe aumento em velocidade e capacidade, e redução no preço. O desempenho é duplicado a cada 18 meses aproximadamente, até por volta de 2005, quando os problemas de dissipação de energia levaram os fabricantes a começar a multiplicação do número de núcleos de CPU e não a velocidade de clock. Espera-se, atualmente, que futuros aumentos de energia venham de um paralelismo maciço, uma convergência curiosa com as propriedades do cérebro.

É claro que existiam dispositivos de cálculo antes do computador eletrônico. As primeiras máquinas automatizadas, datando do século XVII, foram descritas na página 6. A primeira máquina *programável* foi um tear criado em 1805 por Joseph Marie Jacquard (1752-1834), que utilizava cartões perfurados para armazenar instruções relativas ao padrão a ser tecido. Na metade do século XIX, Charles Babbage (1792-1871) projetou duas máquinas, mas não concluiu nenhuma delas. A “máquina diferencial” se destinava a calcular tabelas matemáticas para projetos de engenharia e científicos. Ela foi finalmente construída e se mostrou funcional em 1991 no Science Museum em Londres (Swade, 2000). A “máquina analítica” de Babbage era bem mais ambiciosa: ela incluía memória endereçável, programas armazenados e saltos condicionais, e foi o primeiro artefato capaz de executar computação universal. A colega de Babbage, Ada Lovelace, filha do poeta Lord Byron, talvez tenha sido a primeira programadora do mundo (a linguagem de programação Ada recebeu esse

nome em homenagem a ela). Ela escreveu programas para a máquina analítica não concluída e até mesmo especulou que a máquina poderia jogar xadrez ou compor música.

A IA também tem uma dívida com a área de software da ciência da computação, que forneceu os sistemas operacionais, as linguagens de programação e as ferramentas necessárias para escrever programas modernos (e artigos sobre eles). Porém, essa é uma área em que a dívida foi paga: o trabalho em IA foi pioneiro em muitas ideias que foram aproveitadas posteriormente na ciência da computação em geral, incluindo compartilhamento de tempo, interpretadores interativos, computadores pessoais com janelas e mouse, ambientes de desenvolvimento rápido, tipo de dados de lista ligada, gerenciamento automático de armazenamento e conceitos fundamentais de programação simbólica, funcional, declarativa e orientada a objetos.

1.2.7 Teoria de controle e cibernética

- Como os artefatos podem operar sob seu próprio controle?

Ctesíbio de Alexandria (cerca de 250 a.C.) construiu a primeira máquina autocontrolada: um relógio de água com um regulador que mantinha uma taxa de fluxo constante. Essa invenção mudou a definição do que um artefato poderia fazer. Antes, somente os seres vivos podiam modificar seu comportamento em resposta a mudanças no ambiente. Outros exemplos de sistemas de controle realimentados autorreguláveis incluem o regulador de máquinas a vapor, criado por JamesWatt (1736-1819), e o termostato, criado por Cornelis Drebbel (1572-1633), que também inventou o submarino. A teoria matemática de sistemas realimentados estáveis foi desenvolvida no século XIX.

A figura central na criação daquilo que se conhece hoje como **teoria de controle** foi Norbert Wiener (1894-1964). Wiener foi um matemático brilhante que trabalhou com Bertrand Russell, entre outros, antes de se interessar por sistemas de controle biológico e mecânico e sua conexão com a cognição. Como Craik (que também utilizou sistemas de controle como modelos psicológicos), Wiener e seus colegas Arturo Rosenblueth e Julian Bigelow desafiaram a ortodoxia behaviorista (Rosenblueth *et al.*, 1943). Eles viram o comportamento consciente como o resultado de um mecanismo regulador tentando minimizar o "erro" — a diferença entre o estado atual e o estado objetivo. No final da década de 1940, Wiener, juntamente com Warren McCulloch, Walter Pitts e John von Neumann, organizou uma série de conferências que influenciou os novos modelos matemáticos e computacionais da cognição. O livro de Wiener, *Cybernetics* (1948), tornou-se *best-seller* e despertou o público para a possibilidade de máquinas dotadas de inteligência artificial. Enquanto isso, na Grã-Bretanha, W. Ross Ashby (Ashby, 1940) foi pioneiro em ideias semelhantes. Ashby, Alan Turing, Grey Walter e outros formaram o Ratio Club para "aqueles que tinham as ideias de Wiener antes de surgir o livro de Wiener". *Design for a Brain* (1948, 1952), de Ashby, elaborava a sua ideia de que a mente poderia ser criada com a utilização de mecanismos **homeostáticos** contendo laços de realimentação para atingir comportamento adaptável estável.

A moderna teoria de controle, em especial o ramo conhecido como controle estocástico ótimo, tem como objetivo o projeto de sistemas que maximizam uma **função objetivo** sobre o tempo. Isso corresponde aproximadamente à nossa visão da IA: projetar sistemas que se comportem de maneira

ótima. Então, por que a IA e a teoria de controle são dois campos diferentes, apesar das conexões estreitas entre seus fundadores? A resposta reside no acoplamento estrito entre as técnicas matemáticas familiares aos participantes e os conjuntos de problemas correspondentes que foram incluídos em cada visão do mundo. O cálculo e a álgebra de matrizes, as ferramentas da teoria de controle, eram adequados para sistemas que podem ser descritos por conjuntos fixos de variáveis contínuas, enquanto a IA foi criada em parte como um meio de escapar das limitações percebidas. As ferramentas de inferência lógica e computação permitiram que os pesquisadores da IA considerassem alguns problemas como linguagem, visão e planejamento, que ficavam completamente fora do campo de ação da teoria de controle.

1.2.8 Linguística

- Como a linguagem se relaciona com o pensamento?

Em 1957, B. F. Skinner publicou *Verbal Behavior*. Essa obra foi uma descrição completa e detalhada da abordagem behaviorista para o aprendizado da linguagem, escrita pelo mais proeminente especialista no campo. Porém, curiosamente, uma crítica do livro se tornou tão conhecida quanto o próprio livro e serviu para aniquilar o interesse pelo behaviorismo. O autor da resenha foi o linguista Noam Chomsky, que tinha acabado de publicar um livro sobre sua própria teoria, *Syntactic Structures* (*Estruturas sintáticas*). Chomsky chamou a atenção para o fato de que a teoria behaviorista não tratava da noção de criatividade na linguagem — ela não explicava como uma criança podia compreender e formar frases que nunca tinha ouvido antes. A teoria de Chomsky — baseada em modelos sintáticos criados pelo linguista indiano Panini (c. 350 a.C.) — podia explicar esse fato e, diferentemente das teorias anteriores, era formal o bastante para poder, em princípio, ser programada.

Portanto, a linguística moderna e a IA “nasceram” aproximadamente na mesma época e cresceram juntas, cruzando-se em um campo híbrido chamado **linguística computacional** ou **processamento de linguagem natural**. O problema de compreender a linguagem logo se tornou consideravelmente mais complexo do que parecia em 1957. A compreensão da linguagem exige a compreensão do assunto e do contexto, não apenas a compreensão da estrutura das frases. Isso pode parecer óbvio, mas só foi amplamente avaliado na década de 1960. Grande parte do trabalho anterior em **representação do conhecimento** (o estudo de como colocar o conhecimento em uma forma que um computador possa utilizar) estava vinculado à linguagem e era suprido com informações da pesquisa em linguística que, por sua vez, estava conectada a décadas de pesquisa sobre a análise filosófica da linguagem.

1.3 HISTÓRIA DA INTELIGÊNCIA ARTIFICIAL

Com o material que vimos até agora, estamos prontos para estudar o desenvolvimento da própria IA.

1.3.1 A gestação da inteligência artificial (1943-1955)

O primeiro trabalho agora reconhecido como IA foi realizado por Warren McCulloch e Walter Pitts (1943). Eles se basearam em três fontes: o conhecimento da fisiologia básica e da função dos neurônios no cérebro; uma análise formal da lógica proposicional criada por Russell e Whitehead; e a teoria da computação de Turing. Esses dois pesquisadores propuseram um modelo de neurônios artificiais, no qual cada neurônio se caracteriza por estar "ligado" ou "desligado", com a troca para "ligado" ocorrendo em resposta à estimulação por um número suficiente de neurônios vizinhos. O estado de um neurônio era considerado "equivalente em termos concretos a uma proposição que definia seu estímulo adequado". Por exemplo, eles mostraram que qualquer função computável podia ser calculada por certa rede de neurônios conectados e que todos os conectivos lógicos (e, ou, não etc.) podiam ser implementados por estruturas de redes simples. McCulloch e Pitts também sugeriram que redes definidas adequadamente seriam capazes de aprender. Donald Hebb (1949) demonstrou uma regra de atualização simples para modificar as intensidades de conexão entre neurônios. Sua regra, agora chamada **aprendizado de Hebb**, continua a ser um modelo influente até hoje.

Dois alunos de Harvard, Marvin Minsky e Dean Edmonds, construíram o primeiro computador de rede neural em 1950. O SNARC, como foi chamado, usava 3.000 válvulas eletrônicas e um mecanismo de piloto automático retirado de um bombardeiro B-24 para simular uma rede de 40 neurônios. Mais tarde, em Princeton, Minsky estudou computação universal em redes neurais. A banca examinadora de seu doutorado mostrou-se cética sobre esse tipo de trabalho, sem saber se deveria ser classificado como um trabalho de matemática. Porém, segundo contam, von Neumann teria dito: "Se não é agora, será algum dia." Mais tarde, Minsky acabou provando teoremas importantes que mostravam as limitações da pesquisa em redes neurais.

Surgiram vários exemplos de trabalhos que hoje podem ser caracterizados como IA, mas a visão de Alan Turing foi talvez a mais influente. Já em 1947, ele proferia palestras sobre o tema na Sociedade Matemática de Londres e articulou um programa de trabalhos persuasivo em seu artigo de 1950, "Computing Machinery and Intelligence". Nesse artigo, ele apresentou o teste de Turing, aprendizagem de máquina, algoritmos genéticos e aprendizagem por reforço. Propôs a ideia do *Child Programme*, explicando: "Em vez de tentar produzir um programa para estimular a mente adulta, não seria melhor produzir um que estimulasse a mente infantil?"

1.3.2 O nascimento da inteligência artificial (1956)

Princeton foi o lar de outra figura influente na IA, John McCarthy. Após receber seu PhD lá, em 1951, e trabalhar por dois anos como instrutor, McCarthy mudou-se para Stanford e depois para Dartmouth College, que iria se tornar o local oficial de nascimento desse campo. McCarthy convenceu Minsky, Claude Shannon e Nathaniel Rochester a ajudá-lo a reunir pesquisadores dos Estados Unidos interessados em teoria de autômatos, redes neurais e estudo da inteligência. Eles organizaram um seminário de dois meses em Dartmouth, no verão de 1956. A proposta dizia:[11]

> Propusemos que um estudo de dois meses e dez homens sobre inteligência artificial fosse realizado durante o verão de 1956 no Dartmouth College, em Hanover, New Hampshire. O estudo era para prosseguir com a conjetura básica de que cada aspecto da aprendizagem ou qualquer outra característica da inteligência pode, em princípio, ser descrita tão precisamente a ponto de ser construída uma máquina para simulá-la. Será realizada uma tentativa para descobrir como fazer com que as máquinas usem a linguagem, a partir de abstrações e conceitos, resolvam os tipos de problemas hoje reservados aos seres humanos e se aperfeiçoem. Achamos que poderá haver avanço significativo em um ou mais desses problemas se um grupo cuidadosamente selecionado de cientistas trabalhar em conjunto durante o verão.

Havia 10 participantes ao todo, incluindo Trenchard More, de Princeton, Arthur Samuel, da IBM, e Ray Solomonoff e Oliver Selfridge, do MIT.

Dois pesquisadores da Carnegie Tech,[12] Allen Newell e Herbert Simon, simplesmente roubaram o show. Embora os outros tivessem ideias e, em alguns casos, programas para aplicações específicas como jogos de damas, Newell e Simon já tinham um programa de raciocínio, o Logic Theorist (LT), sobre o qual Simon afirmou: "Criamos um programa de computador capaz de pensar não numericamente e, assim, resolvemos o antigo dilema mente-corpo."[13] Logo após o seminário, o programa foi capaz de demonstrar a maioria dos teoremas do Capítulo 2 do livro *Principia Mathematica* de Russell e Whitehead. Contam que Russell ficou encantado quando Simon mostrou a ele que o programa havia criado uma prova de um teorema que era mais curta que a do livro. Os editores do *Journal of Symbolic Logic* ficaram menos impressionados; eles rejeitaram um artigo escrito em parceria por Newell, Simon e pelo Logic Theorist.

O seminário de Dartmouth não trouxe nenhuma novidade, mas apresentou uns aos outros todos os personagens importantes da história. Nos 20 anos seguintes, o campo seria dominado por essas pessoas e por seus alunos e colegas do MIT, da CMU, de Stanford e da IBM.

Examinando a proposta do seminário de Dartmouth (McCarthy *et al*., 1955), podemos ver por que era necessário que a IA se tornasse um campo separado. Por que todo o trabalho feito na IA não podia ficar sob o nome de teoria de controle, pesquisa operacional ou teoria da decisão que, afinal de contas, têm objetivos semelhantes aos da IA? Ou, então, por que a IA não poderia ser um ramo da matemática? Primeiro, porque a IA abraçou desde o início a ideia de reproduzir faculdades humanas como criatividade, autoaperfeiçoamento e uso da linguagem, e nenhum dos outros campos tratava dessas questões. A segunda resposta é a metodologia. A IA é o único desses campos que claramente é um ramo da ciência da computação (embora a pesquisa operacional compartilhe uma ênfase em simulações por computador), e a IA é o único campo a tentar construir máquinas que funcionarão de forma autônoma em ambientes complexos e mutáveis.

1.3.3 Entusiasmo inicial, grandes expectativas (1952-1969)

Os primeiros anos da IA foram repletos de sucessos, mas de uma forma limitada. Considerando-se os primitivos computadores, as ferramentas de programação da época e o fato de que apenas alguns anos antes os computadores eram vistos como objetos capazes de efetuar operações aritméticas e

nada mais, causava surpresa o fato de um computador realizar qualquer atividade remotamente inteligente. Em geral, a classe intelectual preferia acreditar que “uma máquina nunca poderá realizar *X*” (veja, no Capítulo 26, uma longa lista de *X* reunidos por Turing). Os pesquisadores da IA respondiam naturalmente demonstrando um *X* após outro. John McCarthy se referiu a esse período como a era do “Olhe, mamãe, sem as mãos!”.

O sucesso inicial de Newell e Simon prosseguiu com o General Problem Solver (solucionador de problemas gerais) ou GPS. Diferentemente do Logic Theorist, esse programa foi projetado desde o início para imitar protocolos humanos de resolução de problemas. Dentro da classe limitada de quebra-cabeças com a qual podia lidar, verificou-se que a ordem em que o programa considerava submetas e ações possíveis era semelhante à ordem em que os seres humanos abordavam os mesmos problemas. Desse modo, o GPS talvez tenha sido o primeiro programa a incorporar a abordagem de “pensar de forma humana”. O sucesso do GPS e de programas subsequentes como modelos de cognição levou Newell e Simon (1976) a formularem a famosa hipótese do **sistema de símbolos físicos**, que afirma que “um sistema de símbolos físicos tem os meios necessários e suficientes para uma ação inteligente geral”. O que eles queriam dizer é que qualquer sistema (ser humano ou máquina) que exiba inteligência deve operar manipulando estruturas de dados compostas por símbolos. Veremos, mais adiante, que essa hipótese enfrentou desafios provenientes de muitas direções.

Na IBM, Nathaniel Rochester e seus colegas produziram alguns dos primeiros programas de IA. Herbert Gelernter (1959) construiu o Geometry Theorem Prover, que podia demonstrar teoremas que seriam considerados bastante complicados por muitos alunos de matemática. A partir de 1952, Arthur Samuel escreveu uma série de programas para jogos de damas que eventualmente aprendiam a jogar em um nível amador elevado. Ao mesmo tempo, ele contestou a ideia de que os computadores só podem realizar as atividades para as quais foram programados: seu programa aprendeu rapidamente a jogar melhor que seu criador. O programa foi demonstrado na televisão em fevereiro de 1956, causando impressão muito forte. Como Turing, Samuel teve dificuldades para conseguir um horário em que pudesse utilizar os computadores. Trabalhando à noite, ele usou máquinas que ainda estavam na bancada de testes na fábrica da IBM. O Capítulo 5 aborda os jogos de computador, e o Capítulo 21 explica as técnicas de aprendizado usadas por Samuel.

John McCarthy saiu de Dartmouth para o MIT e lá contribuiu com três realizações cruciais em um ano histórico: 1958. No MIT AI Lab Memo Nº. 1, McCarthy definiu a linguagem de alto nível **Lisp**, que acabou por se tornar a linguagem de programação dominante na IA pelos próximos 30 anos. Com o Lisp, McCarthy teve a ferramenta de que precisava, mas o acesso a recursos de computação escassos e dispendiosos também era um sério problema. Em resposta, ele e outros pesquisadores do MIT criaram o compartilhamento de tempo (*time sharing*). Também em 1958, McCarthy publicou um artigo intitulado *Programs with common sense*, em que descrevia o Advice Taker, um programa hipotético que pode ser visto como o primeiro sistema de IA completo. Como o Logic Theorist e o Geometry Theorem Prover, o programa de McCarthy foi projetado para usar o conhecimento com a finalidade de buscar soluções para problemas.

Entretanto, diferentemente dos outros, ele procurava incorporar o conhecimento geral do mundo. Por exemplo, McCarthy mostrou que alguns axiomas simples permitiriam ao programa gerar um plano para dirigir até o aeroporto e embarcar em um avião. O programa também foi criado de forma

a poder aceitar novos axiomas no curso normal de operação, permitindo assim que adquirisse competência em novas áreas *sem ser reprogramado*. Portanto, o Advice Taker incorporava os princípios centrais de representação de conhecimento e de raciocínio: de que é útil ter uma representação formal e explícita do mundo do modo como as ações de um agente afetam o mundo e o seu funcionamento, e ser capaz de manipular essa representação com processos dedutivos. É notável como grande parte do artigo de 1958 permanece relevante até hoje.

O ano de 1958 também marcou a época em que Marvin Minsky foi para o MIT. Porém, sua colaboração inicial com McCarthy não durou muito. McCarthy enfatizava a representação e o raciocínio em lógica formal, enquanto Minsky estava mais interessado em fazer os programas funcionarem e, eventualmente, desenvolveu uma perspectiva contrária ao estudo da lógica. Em 1963, McCarthy fundou o laboratório de IA em Stanford. Seu plano de usar a lógica para construir o Advice Taker definitivo foi antecipado pela descoberta feita por J. A. Robinson do método de resolução (um algoritmo completo para demonstração de teoremas para a lógica de primeira ordem; consulte o Capítulo 9). O trabalho em Stanford enfatizou métodos de uso geral para raciocínio lógico. As aplicações da lógica incluíam os sistemas para responder a perguntas e os sistemas de planejamento de Cordell Green (Green, 1969b) e o projeto de robótica Shakey no novo Stanford Research Institute (SRI). Este último projeto, descrito com mais detalhes no Capítulo 25, foi o primeiro a demonstrar a integração completa do raciocínio lógico e da atividade física.

Minsky orientou vários alunos que escolheram problemas limitados cuja solução parecia exigir inteligência. Esses domínios limitados se tornaram conhecidos como **micromundos**. O programa SAINT de James Slagle (1963) era capaz de resolver problemas de cálculo integral típicos do primeiro ano dos cursos acadêmicos. O programa ANALOGY de Tom Evans (1968) resolvia problemas de analogia geométrica que apareciam em testes de QI. O programa STUDENT de Daniel Bobrow (1967) resolvia problemas clássicos de álgebra, como este:

> Se o número de clientes que Tom consegue é igual ao dobro do quadrado de 20% do número de anúncios que ele publica e se o número de anúncios publicados é 45, qual é o número de clientes que Tom consegue?

O mais famoso micromundo foi o mundo de blocos, que consiste em um conjunto de blocos sólidos colocados sobre uma mesa (ou, com maior frequência, sobre a simulação de uma mesa), como mostra a Figura 1.4. Uma tarefa típica nesse mundo é reorganizar os blocos de certa maneira, utilizando a mão de um robô que pode erguer um bloco de cada vez. O mundo de blocos foi a base do projeto de visão de David Huffman (1971), do trabalho em visão e propagação de restrições de David Waltz (1975), da teoria do aprendizado de Patrick Winston (1970), do programa de compreensão de linguagem natural de Terry Winograd (1972) e do sistema de planejamento de Scott Fahlman (1974).

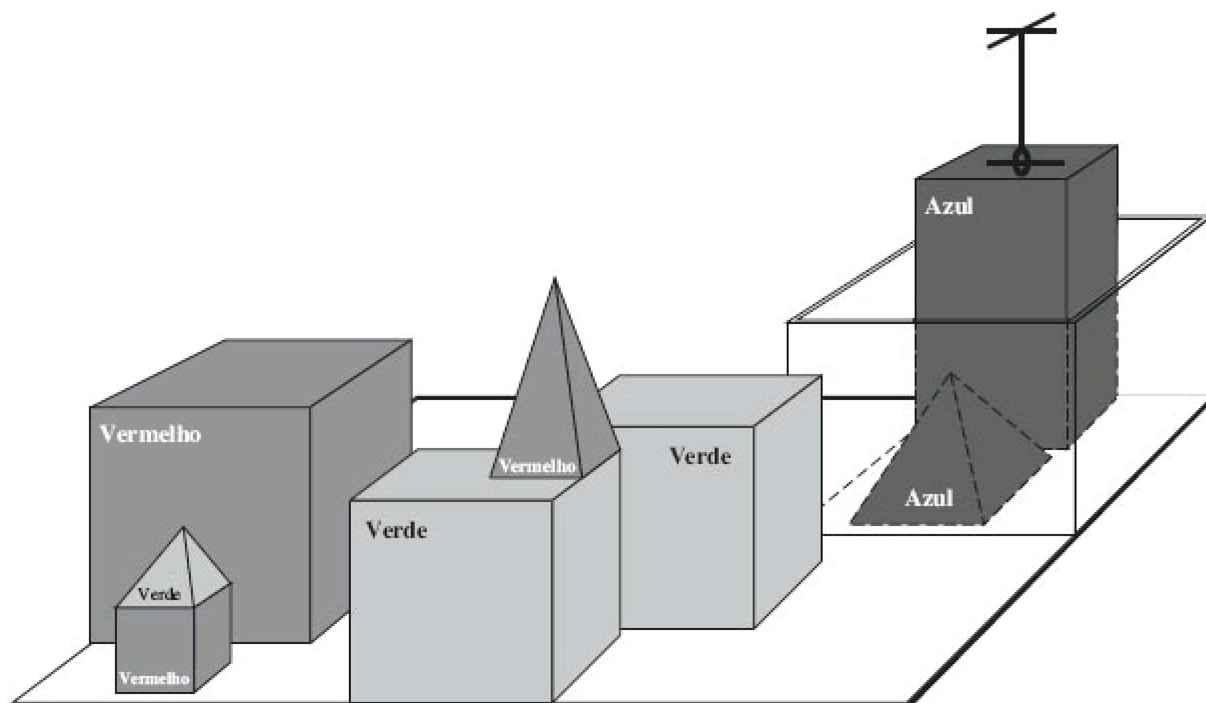


Figura 1.4 Uma cena do mundo de blocos. O programa SHRDLU (Winograd, 1972) tinha acabado de completar o comando: “Encontre um bloco mais alto que o bloco que você está segurando e coloque-o na caixa.”

O trabalho pioneiro baseado nas redes neurais de McCulloch e Pitts também prosperou. O trabalho de Winograd e Cowan (1963) mostrou que grande número de elementos podia representar coletivamente um conceito individual, com aumento correspondente na robustez e no paralelismo. Os métodos de aprendizado de Hebb foram aperfeiçoados por Bernie Widrow (Widrow e Hoff, 1960; Widrow, 1962), que denominou suas redes **adalines**, e por Frank Rosenblatt (1962) com seus **perceptrons**. **O teorema da convergência do perceptron** (Block *et al.,* 1962) determina que o algoritmo de aprendizagem podia ajustar os pesos de conexão de um perceptron para corresponderem a quaisquer dados de entrada, desde que existisse tal correspondência. Esses tópicos são cobertos no Capítulo 20.

1.3.4 Uma dose de realidade (1966-1973)

Desde o início, os pesquisadores da IA eram ousados nos prognósticos de seus sucessos futuros. Esta declaração de Herbert Simon em 1957 frequentemente é citada:

> Não é meu objetivo surpreendê-los ou chocá-los, mas o modo mais simples de resumir tudo isso é dizer que agora existem no mundo máquinas que pensam, aprendem e criam. Além disso, sua capacidade de realizar essas atividades está crescendo rapidamente até o ponto — em um futuro visível — no qual a variedade de problemas com que elas poderão lidar será correspondente à variedade de problemas com os quais lida a mente humana.

Termos como “futuro visível” podem ser interpretados de várias maneiras, mas Simon também fez predições mais concretas: a de que dentro de 10 anos um computador seria campeão de xadrez e que um teorema matemático significativo seria provado por uma máquina. Essas previsões se realizaram

(ou quase) no prazo de 40 anos, em vez de 10. O excesso de confiança de Simon se devia ao desempenho promissor dos primeiros sistemas de IA em exemplos simples. Contudo, em quase todos os casos, esses primeiros sistemas acabaram falhando desastrosamente quando foram experimentados em conjuntos de problemas mais extensos ou em problemas mais difíceis.

O primeiro tipo de dificuldade surgiu porque a maioria dos primeiros programas não tinha conhecimento de seu assunto; eles obtinham sucesso por meio de manipulações sintáticas simples. Uma história típica ocorreu durante os primeiros esforços de tradução automática, que foram generosamente subsidiados pelo National Research Council dos Estados Unidos, em uma tentativa de acelerar a tradução de documentos científicos russos após o lançamento do Sputnik em 1957. Inicialmente, imaginava-se que transformações sintáticas simples baseadas nas gramáticas russas e inglesas, e a substituição de palavras com a utilização de um dicionário eletrônico, seriam suficientes para preservar os significados exatos das orações. O fato é que a tradução exata exige conhecimento profundo do assunto para solucionar ambiguidades e estabelecer o conteúdo da sentença. A famosa retradução de "o espírito está disposto mas a carne é fraca"[14] como "a vodca é boa mas a carne é podre" ilustra as dificuldades encontradas. Em 1966, um relatório criado por um comitê consultivo descobriu que "não existe nenhum sistema de tradução automática para texto científico em geral, e não existe nenhuma perspectiva imediata nesse sentido". Toda a subvenção do governo dos Estados Unidos para projetos acadêmicos de tradução foi cancelada. Hoje, a tradução automática é uma ferramenta imperfeita, mas amplamente utilizada em documentos técnicos, comerciais, governamentais e da Internet.

O segundo tipo de dificuldade foi a impossibilidade de tratar muitos dos problemas que a IA estava tentando resolver. A maior parte dos primeiros programas de IA resolvia problemas experimentando diferentes combinações de passos até encontrar a solução. Essa estratégia funcionou inicialmente porque os micromundos continham pouquíssimos objetos e, consequentemente, um número muito pequeno de ações possíveis e sequências de soluções muito curtas. Antes do desenvolvimento da teoria de complexidade computacional, era crença geral que o "aumento da escala" para problemas maiores era apenas uma questão de haver hardware mais rápido e maior capacidade de memória. Por exemplo, o otimismo que acompanhou o desenvolvimento da prova de teoremas por resolução logo foi ofuscado quando os pesquisadores não conseguiram provar teoremas que envolviam mais que algumas dezenas de fatos. *O fato de um programa poder encontrar uma solução em princípio não significa que o programa contenha quaisquer dos mecanismos necessários para encontrá-la na prática.*

A ilusão do poder computacional ilimitado não ficou confinada aos programas de resolução de problemas. Os primeiros experimentos de **evolução automática** (agora chamados **algoritmos genéticos**) (Friedberg, 1958; Friedberg *et al.*, 1959) se baseavam na convicção sem dúvida correta de que, realizando-se uma série apropriada de pequenas mutações em um programa em código de máquina, seria possível gerar um programa com bom desempenho para qualquer tarefa simples. Então, a ideia era experimentar mutações aleatórias com um processo de seleção para preservar mutações que parecessem úteis. Apesar de milhares de horas de tempo de CPU, quase nenhum progresso foi demonstrado. Os algoritmos genéticos modernos utilizam representações melhores e têm mais sucesso.

A incapacidade de conviver com a "explosão combinatória" foi uma das principais críticas à IA

contidas no relatório Lighthill (Lighthill, 1973), que formou a base para a decisão do governo britânico de encerrar o apoio à pesquisa da IA em todas as universidades, com exceção de duas (a tradição oral pinta um quadro um pouco diferente e mais colorido, com ambições políticas e hostilidades pessoais, cuja descrição não nos interessa aqui).

Uma terceira dificuldade surgiu devido a algumas limitações fundamentais nas estruturas básicas que estavam sendo utilizadas para gerar o comportamento inteligente. Por exemplo, o livro de Minsky e Papert, *Perceptrons* (1969), provou que, embora os perceptrons (uma forma simples de rede neural) pudessem aprender tudo o que eram capazes de representar, eles podiam representar muito pouco. Em particular, um perceptron de duas entradas (restringido para ser mais simples que a forma que Rosemblatt estudou) não podia ser treinado para reconhecer quando suas duas entradas eram diferentes. Embora seus resultados não se aplicassem a redes mais complexas de várias camadas, a subvenção de pesquisas relacionadas a redes neurais logo se reduziu a quase nada. Ironicamente, os novos algoritmos de aprendizado por retropropagação para redes de várias camadas que acabaram de provocar um enorme renascimento na pesquisa de redes neurais no final da década de 1980 foram, na verdade, descobertos primeiro em 1969 (Bryson e Ho, 1969).

1.3.5 Sistemas baseados em conhecimento: a chave para o poder? (1969-1979)

O quadro de resolução de problemas que havia surgido durante a primeira década de pesquisas em IA foi o de um mecanismo de busca de uso geral que procurava reunir passos elementares de raciocínio para encontrar soluções completas. Tais abordagens foram chamadas **métodos fracos** porque, embora gerais, não podiam ter aumento de escala para instâncias de problemas grandes ou difíceis. A alternativa para métodos fracos é usar um conhecimento mais amplo e específico de domínio que permita passos de raciocínio maiores e que possam tratar com mais facilidade casos que ocorrem tipicamente em especialidades estritas. Podemos dizer que, para resolver um problema difícil, praticamente é necessário já saber a resposta.

O programa DENDRAL (Buchanan *et al*., 1969) foi um exemplo inicial dessa abordagem. Ele foi desenvolvido em Stanford, onde Ed Feigenbaum (um antigo aluno de Herbert Simon), Bruce Buchanan (filósofo transformado em cientista de computação) e Joshua Lederberg (geneticista laureado com um Prêmio Nobel) formaram uma equipe para resolver o problema de inferir a estrutura molecular a partir das informações fornecidas por um espectrômetro de massa. A entrada para o programa consiste na fórmula elementar da molécula (por exemplo, $C_6H_{13}NO_2$) e o espectro de massa que fornece as massas dos diversos fragmentos da molécula gerada quando ela é bombardeada por um feixe de elétrons. Por exemplo, o espectro de massa poderia conter um pico em $m = 15$, correspondendo à massa de um fragmento metil (CH_3).

A versão ingênua do programa gerou todas as estruturas possíveis consistentes com a fórmula e depois previu qual seria o espectro de massa observado para cada uma, comparando esse espectro com o espectro real. Como se poderia esperar, esse é um problema intratável mesmo para moléculas de tamanho moderado. Os pesquisadores do DENDRAL consultaram especialistas em química analítica e descobriram que eles trabalhavam procurando padrões conhecidos de picos no espectro que sugerissem subestruturas comuns na molécula. Por exemplo, a regra a seguir é usada para

reconhecer um subgrupo cetona (C=O), que pesa 28 unidades de massa:

> **se** existem dois picos em x_1 e x_2 tais que
> (a) $x_1 + x_2 = M + 28$ (M é a massa da molécula inteira);
> (b) $x_1 - 28$ é um pico;
> (c) $x_2 - 28$ é um pico;
> (d) No mínimo, um entre x_1 e x_2 é alto.
> **então**, existe um subgrupo cetona

O reconhecimento de que a molécula contém uma subestrutura específica reduz enormemente o número de possíveis candidatos. O DENDRAL era eficiente porque:

> Todo o conhecimento teórico relevante para resolver esses problemas foi mapeado de sua forma geral no [componente de previsão de espectro] ("princípios básicos") para formas especiais eficientes ("receitas de bolo"). (Feigenbaum *et al*., 1971)

O DENDRAL foi importante porque representou o primeiro sistema bem-sucedido de *conhecimento intensivo*: sua habilidade derivava de um grande número de regras de propósito específico. Sistemas posteriores também incorporaram o tema principal da abordagem de McCarthy no Advice Taker — a separação clara entre o conhecimento (na forma de regras) e o componente de raciocínio.

Com essa lição em mente, Feigenbaum e outros pesquisadores de Stanford iniciaram o Heuristic Programming Project (HPP) para investigar até que ponto a nova metodologia de **sistemas especialistas** poderia ser aplicada a outras áreas do conhecimento humano. Em seguida, o principal esforço foi dedicado à área de diagnóstico médico. Feigenbaum, Buchanan e o Dr. Edward Shortliffe desenvolveram o MYCIN para diagnosticar infecções sanguíneas. Com cerca de 450 regras, o MYCIN foi capaz de se sair tão bem quanto alguns especialistas e muito melhor do que médicos em início de carreira. Ele também apresentava duas diferenças importantes em relação ao DENDRAL. Primeiro, diferentemente das regras do DENDRAL, não havia nenhum modelo teórico geral a partir do qual as regras do MYCIN pudessem ser deduzidas. Elas tinham de ser adquiridas a partir de entrevistas extensivas com especialistas que, por sua vez, as adquiriam de livros didáticos, de outros especialistas e da experiência direta de estudos de casos. Em segundo lugar, as regras tinham de refletir a incerteza associada ao conhecimento médico. O MYCIN incorporava um cálculo de incerteza chamado **fatores de certeza** (consulte o Capítulo 14) que pareciam (na época) se adequar bem à forma como os médicos avaliavam o impacto das evidências no diagnóstico.

A importância do conhecimento de domínio também ficou aparente na área da compreensão da linguagem natural. Embora o sistema SHRDLU de Winograd para reconhecimento da linguagem natural tivesse despertado bastante interesse, sua dependência da análise sintática provocou alguns problemas idênticos aos que ocorreram nos primeiros trabalhos em tradução automática. Ele foi capaz de superar a ambiguidade e reconhecer referências pronominais, mas isso acontecia principalmente porque o programa foi criado especificamente para uma única área — o mundo dos blocos. Diversos pesquisadores, entre eles Eugene Charniak, aluno graduado e companheiro de

Winograd no MIT, sugeriram que uma compreensão robusta da linguagem exigiria conhecimentos gerais sobre o mundo e um método genérico para utilizar esses conhecimentos.

Em Yale, o linguista transformado em pesquisador da IA Roger Schank enfatizou esse ponto, afirmando: "Não existe essa coisa de sintaxe." Isso irritou muitos linguistas, mas serviu para dar início a uma discussão útil. Schank e seus alunos elaboraram uma série de programas (Schank e Abelson, 1977; Wilensky, 1978; Schank e Riesbeck, 1981; Dyer, 1983), todos com a tarefa de entender a linguagem natural. Porém, a ênfase foi menos na linguagem em si e mais nos problemas de representação e raciocínio com o conhecimento exigido para compreensão da linguagem. Os problemas incluíam a representação de situações estereotípicas (Cullingford, 1981), descrição da organização da memória humana (Rieger, 1976; Kolodner, 1983) e compreensão de planos e metas (Wilensky, 1983).

O enorme crescimento das aplicações para resolução de problemas reais causou um aumento simultâneo na demanda por esquemas utilizáveis de representação do conhecimento. Foi desenvolvido grande número de diferentes linguagens de representação e raciocínio. Algumas se baseavam na lógica — por exemplo, a linguagem Prolog se tornou popular na Europa, e a família PLANNER, nos Estados Unidos. Outras, seguindo a ideia de **frames** de Minsky (1975), adotaram uma abordagem mais estruturada, reunindo fatos sobre tipos específicos de objetos e eventos, e organizando os tipos em uma grande hierarquia taxonômica análoga a uma taxonomia biológica.

1.3.6 A IA se torna uma indústria (de 1980 até a atualidade)

O primeiro sistema especialista comercial bem-sucedido, o R1, iniciou sua operação na Digital Equipment Corporation (McDermott, 1982). O programa ajudou a configurar pedidos de novos sistemas de computadores; em 1986, ele estava fazendo a empresa economizar cerca de 40 milhões de dólares por ano. Em 1988, o grupo de IA da DEC tinha 40 sistemas especialistas entregues, com outros sendo produzidos. A Du Pont tinha 100 desses sistemas em uso e 500 em desenvolvimento, economizando aproximadamente 10 milhões de dólares por ano. Quase todas as corporações importantes dos Estados Unidos tinham seu próprio grupo de IA e estavam usando ou investigando sistemas especialistas.

Em 1981, os japoneses anunciaram o projeto "Fifth Generation", um plano de 10 anos para montar computadores inteligentes que utilizassem Prolog. Em resposta, os Estados Unidos formaram a Microelectronics and Computer Technology Corporation (MCC) como um consórcio de pesquisa projetado para assegurar a competitividade nacional. Em ambos os casos, a IA fazia parte de um amplo esforço, incluindo o projeto de chips e a pesquisa da interface humana. Na Inglaterra, o relatório Alvey reabilitou o subsídio que havia sido cortado em consequência do relatório Lighthill.[15] No entanto, em todos os três países, os projetos nunca alcançaram seus objetivos ambiciosos.

De modo geral, a indústria da IA se expandiu de alguns milhões de dólares em 1980 para bilhões de dólares em 1988, incluindo centenas de empresas construindo sistemas especialistas, sistemas de visão, robôs, e software e hardware especializados para esses propósitos. Logo depois, veio um período chamado de "inverno da IA", em que muitas empresas caíram no esquecimento à medida que

deixaram de cumprir promessas extravagantes.

1.3.7 O retorno das redes neurais (de 1986 até a atualidade)

Em meados dos anos 1980, pelo menos quatro grupos diferentes reinventaram o algoritmo de aprendizado por **retroprogramação**, descoberto pela primeira vez em 1969 por Bryson e Ho. O algoritmo foi aplicado a muitos problemas de aprendizado em ciência da computação e psicologia, e a ampla disseminação dos resultados na coletânea *Parallel Distributed Processing* (Rumelhart e McClelland, 1986) causou grande excitação.

Os chamados modelos **conexionistas** para sistemas inteligentes eram vistos por alguns como concorrentes diretos dos modelos simbólicos promovidos por Newell e Simon e da abordagem logicista de McCarthy e outros pesquisadores (Smolensky, 1988). Pode parecer óbvio que, em certo nível, os seres humanos manipulam símbolos — de fato, o livro de Terrence Deacon, *The Symbolic Species* (1997), sugere que essa é a *característica que define* os seres humanos —, mas os conexionistas mais fervorosos questionavam se a manipulação de símbolos tinha qualquer função explicativa real em modelos detalhados de cognição. Essa pergunta permanece sem resposta, mas a visão atual é de que as abordagens conexionista e simbólica são complementares, e não concorrentes. Como ocorreu com a separação da IA e da ciência cognitiva, a pesquisa moderna de rede neural se bifurcou em dois campos, um preocupado com a criação de algoritmos e arquiteturas de rede eficazes e a compreensão de suas propriedades matemáticas, o outro preocupado com a modelagem cuidadosa das propriedades empíricas de neurônios reais e conjuntos de neurônios.

1.3.8 A IA se torna uma ciência (de 1987 até a atualidade)

Nos últimos anos, houve uma revolução no trabalho em inteligência artificial, tanto no conteúdo quanto na metodologia.[16] Agora, é mais comum usar as teorias existentes como bases, em vez de propor teorias inteiramente novas, fundamentar as afirmações em teoremas rigorosos ou na evidência experimental rígida, em vez de utilizar como base a intuição e destacar a relevância de aplicações reais em vez de exemplos de brinquedos.

Em parte, a IA surgiu como uma rebelião contra as limitações de áreas existentes como a teoria de controle e a estatística, mas agora ela inclui esses campos. Conforme afirmou David McAllester (1998):

> No período inicial da IA, parecia plausível que novas formas de computação simbólica, como frames e redes semânticas, tornariam obsoleta grande parte da teoria clássica. Isso levou a uma forma de isolacionismo na qual a IA ficou bem separada do restante da ciência da computação. Atualmente, esse isolacionismo está sendo abandonado. Existe o reconhecimento de que o aprendizado da máquina não deve ser isolado da teoria da informação, de que o raciocínio incerto não deve ser isolado da modelagem estocástica, de que a busca não deve ser isolada da otimização clássica e do controle, e de que o raciocínio automatizado não deve ser isolado dos

métodos formais e da análise estática.

Em termos de metodologia, a IA finalmente adotou com firmeza o método científico. Para serem aceitas, as hipóteses devem ser submetidas a rigorosos experimentos empíricos, e os resultados devem ser analisados estatisticamente de acordo com sua importância (Cohen, 1995). Agora é possível replicar experimentos a partir da utilização de repositórios compartilhados de código e dados de teste.

O campo de reconhecimento da fala ilustra o padrão. Nos anos 1970, foi experimentada ampla variedade de arquiteturas e abordagens diferentes. Muitas delas eram bastante *ad hoc* e frágeis, e foram demonstradas em apenas alguns exemplos especialmente selecionados. Nos últimos anos, abordagens baseadas em **modelos ocultos de Markov** (MOMs) passaram a dominar a área. Dois aspectos dos MOMs são relevantes. Primeiro, eles se baseiam em uma teoria matemática rigorosa. Isso permitiu que os cientistas de reconhecimento de fala se baseassem em várias décadas de resultados matemáticos desenvolvidos em outros campos. Em segundo lugar, eles são gerados por um processo de treinamento em um grande conjunto de dados reais de fala. Isso assegura um desempenho robusto e, em testes cegos rigorosos, os MOMs têm melhorado suas pontuações de forma contínua. A tecnologia da fala e o campo inter-relacionado de reconhecimento de caracteres manuscritos já estão efetuando a transição para aplicações industriais e de consumo em larga escala.

Observe que não há nenhuma afirmação científica de que os humanos utilizam MOMs para reconhecer a fala, mas que os MOMs fornecem uma estrutura matemática para a compreensão do problema e apoiam a alegação da engenharia de que na prática eles funcionam bem.

A tradução automática segue o mesmo curso que o reconhecimento de voz. Na década de 1950 houve um entusiasmo inicial por uma abordagem baseada na sequência de palavras, aprendida com modelos de acordo com os princípios da teoria da informação. A abordagem caiu em desuso na década de 1960, mas retornou no final dos anos 1990 e agora domina o campo.

As redes neurais também seguem essa tendência. Grande parte do trabalho em redes neurais nos anos 1980 foi realizada na tentativa de definir a abrangência do que poderia ser feito e de aprender como as redes neurais diferem das técnicas "tradicionais". Utilizando metodologia aperfeiçoada e estruturas teóricas, o campo chegou a uma compreensão tal que, agora, as redes neurais podem ser comparadas a técnicas correspondentes da estatística, do reconhecimento de padrões e do aprendizado de máquina, podendo ser utilizada a técnica mais promissora em cada aplicação. Como resultado desse desenvolvimento, a tecnologia denominada **mineração de dados** gerou uma nova e vigorosa indústria.

A obra de Judea Pearl, *Probabilistic Reasoning in Intelligent Systems* (1988), levou a uma nova aceitação da probabilidade e da teoria da decisão na IA, seguindo um renascimento do interesse descrito no artigo de Peter Cheeseman, "In Defense of Probability" (1985). O formalismo denominado **rede bayesiana** foi criado para permitir a representação eficiente do conhecimento incerto e o raciocínio rigoroso com a utilização desse tipo de conhecimento. Essa abordagem supera amplamente muitos problemas dos sistemas de raciocínio probabilístico das décadas de 1960 e 1970; agora ele domina a pesquisa de IA sobre raciocínio incerto e sistemas especialistas.

A abordagem admite o aprendizado a partir da experiência e combina o melhor da IA clássica e das redes neurais. O trabalho de Judea Pearl (1982a) e de Eric Horvitz e David Heckerman (Horvitz

e Heckerman, 1986; Horvitz *et al*., 1986) promoveu a ideia de sistemas especialistas *normativos*: sistemas que agem racionalmente de acordo com as leis da teoria de decisão e não procuram imitar os passos do pensamento de especialistas humanos. O sistema operacional Windows™ inclui vários sistemas especialistas de diagnóstico normativo para correção de problemas. Os Capítulos 13 a 16 examinam essa área.

Revoluções suaves semelhantes a essa ocorreram nos campos de robótica, visão computacional e representação de conhecimento. Uma compreensão melhor dos problemas e de suas propriedades de complexidade, combinada à maior sofisticação matemática, resultou em agendas de pesquisa utilizáveis e métodos robustos. Apesar do aumento da formalização e da especialização terem levado campos como visão e robótica a tornarem-se de alguma forma isolados do "principal" em IA nos anos 1990, essa tendência foi revertida nos últimos anos à medida que ferramentas de aprendizado de máquina em particular, mostraram-se eficazes para muitos problemas. O processo de reintegração já está rendendo benefícios significativos.

1.3.9 O surgimento de agentes inteligentes (de 1995 até a atualidade)

Talvez encorajados pelo progresso na resolução dos subproblemas da IA, os pesquisadores também começaram a examinar mais uma vez o problema do "agente como um todo". O trabalho de Allen Newell, John Laird e Paul Rosenbloom no SOAR (Newell, 1990; Laird *et al*., 1987) é o exemplo mais conhecido de uma arquitetura completa de agente. Um dos ambientes mais importantes para agentes inteligentes é a Internet. Os sistemas de IA se tornaram tão comuns em aplicações da Web que o sufixo "bot" passou a fazer parte da linguagem cotidiana. Além disso, as tecnologias da IA servem de base a muitas ferramentas da Internet, como mecanismos de pesquisa, sistemas de recomendação (*recommender systems*) e agregadores de conteúdo de construção de sites.

Uma consequência de tentar construir agentes completos é a constatação de que os subcampos previamente isolados da IA podem necessitar ser reorganizados quando se tiver que unir os resultados. Em particular, hoje é amplamente reconhecido que os sistemas sensoriais (visão, sonar, reconhecimento de voz etc.) não podem fornecer informações perfeitamente confiáveis sobre o meio ambiente. Assim, os sistemas de raciocínio e de planejamento devem ser capazes de lidar com a incerteza. Uma segunda consequência importante pela perspectiva do agente é que a IA foi estabelecida em contato muito mais próximo com outros campos, como teoria de controle e economia, que também lidam com agentes. O progresso recente do controle de carros robóticos foi derivado de uma mistura de abordagens que vai desde melhores sensores, controle teórico da integração do sensoriamento, localização e mapeamento, bem como um grau de alto nível de planejamento.

Apesar desses sucessos, alguns fundadores influentes da IA, incluindo John McCarthy (2007), Marvin Minsky (2007), Nils Nilsson (1995, 2005) e Patrick Winston (Beal e Winston, 2009), expressaram descontentamento com a evolução da IA. Achavam que a IA deveria colocar menos ênfase na criação de versões cada vez melhores de aplicações eficientes para tarefas específicas, tal como dirigir um carro, jogar xadrez ou reconhecer fala. Em vez disso, acreditam que a IA deveria retornar às suas raízes esforçando-se para obter, nas palavras de Simon, "máquinas que pensam, que

aprendem e que criam". Chamam o esforço de **IA de nível humano** ou HLAI; o primeiro simpósio foi em 2004 (Minsky *et al.,* 2004). O esforço necessitará de grandes bases de conhecimento; Hendler *et al.* (1995) discutem de onde essas bases de conhecimento poderiam vir.

Uma ideia relacionada é o subcampo da **inteligência geral artificial** ou IAG (Goertzel e Pennachin, 2007), que realizou a sua primeira conferência e organizou o *Journal of Artificial General Intelligence* em 2008. A IAG procura por um algoritmo universal para aprender e atuar em qualquer ambiente, e tem suas raízes na obra de Ray Solomonoff (1964), um dos participantes da conferência original de Dartmouth em 1956. Garantindo que o que nós criamos é realmente **IA amigável** também é uma preocupação (Yudkowsky, 2008; Omohundro, 2008), para a qual voltaremos no Capítulo 26.

1.3.10 Disponibilidade de conjuntos de dados muito grandes (2001 até a atualidade)

Ao longo de 60 anos de história da ciência da computação, a ênfase tem sido no *algoritmo* como o assunto principal de estudo. Mas alguns trabalhos recentes da IA sugerem que, para muitos problemas, faz mais sentido se preocupar com os *dados* e ser menos exigente sobre qual algoritmo aplicar. Isso é verdade devido à disponibilidade crescente de fontes de dados muito grandes: por exemplo, trilhões de palavras de inglês e bilhões de imagens da Web (Kilgarriff e Grefenstette, 2006) ou bilhões de pares de bases de sequências genômicas (Collins *et al.,* 2003).

Um artigo influente nessa linha de pesquisa foi o trabalho de Yarowsky (1995) sobre desambiguação de sentido de palavras: dado o uso da palavra "planta" em uma frase, ela se refere a flora ou fábrica? Abordagens anteriores do problema confiavam em rótulos humanos combinados com algoritmos de aprendizado de máquina. Yarowsky mostrou que a tarefa poderia ser feita, com precisão superior a 96%, sem quaisquer exemplos rotulados. Em vez disso, dado um *corpus* muito grande de texto não anotado e apenas as definições de dicionário dos dois sentidos, "obras, planta industrial" e "flora, vida das plantas", pode-se rotular exemplos no *corpus*, e de lá, **por iniciativa própria**, aprender novos modelos que ajudem a rotular novos exemplos. Banko e Brill (2001) mostram que técnicas como essa têm um desempenho ainda melhor à medida que a quantidade de texto disponível vai de um milhão de palavras para um bilhão e que o aumento no desempenho pela utilização de mais dados excede qualquer diferença na escolha do algoritmo; um algoritmo medíocre com 100 milhões de palavras de dados de treinamento não rotulados supera o melhor algoritmo conhecido com um milhão de palavras.

Em outro exemplo, Hays e Efros (2007) discutem o problema do preenchimento de buracos em uma fotografia. Suponha que você use o Photoshop para mascarar um ex-amigo de uma foto de grupo, mas agora você precisa preencher a área mascarada com algo que corresponda ao fundo. Hays e Efros definiram um algoritmo que busca, através de uma coleção de fotos, encontrar algo que vá corresponder. Descobriram que o desempenho de seu algoritmo era pobre quando usavam uma coleção de apenas 10 mil fotos, mas atravessou o limiar para um excelente desempenho quando aumentaram a coleção para dois milhões de fotos.

Trabalho como esse sugere que o "gargalo do conhecimento" na IA — o problema de como

expressar todo o conhecimento que um sistema necessita — pode ser resolvido em muitas aplicações por métodos de aprendizagem, em vez de engenharia do conhecimento codificada à mão, desde que os algoritmos de aprendizado tenham dados suficientes para prosseguir (Halevy *et al*., 2009). Os observadores notaram o surgimento de novas aplicações e escreveram que "o inverno da IA" pode estar produzindo uma nova primavera (Havenstein, 2005). Como Kurzweil (2005) escreveu: "Hoje, muitos milhares de aplicações de IA estão profundamente enraizadas na infraestrutura de cada indústria."

1.4 O ESTADO DA ARTE

O que a IA pode fazer hoje? É difícil uma resposta concisa porque existem muitas atividades em vários subcampos. Aqui, mostramos algumas aplicações; outras serão apresentadas ao longo do livro.

Veículos robóticos: Um carro robótico sem motorista chamado STANLEY acelerou através do terreno acidentado do deserto Mojave a 22 mph, terminando o percurso de 212 quilômetros como o primeiro para ganhar o DARPA Grand Challenge 2005. STANLEY é um Touareg Volkswagen equipado com câmeras, radares e telêmetros a *laser* para detectar o ambiente e computador de bordo para comandar a pilotagem, a frenagem e a aceleração (Thrun, 2006). No ano seguinte, o BOSS da CMU ganhou o Urban Chalenge, dirigindo de forma segura no trânsito pelas ruas de uma base da força aérea fechada, obedecendo às regras de trânsito e evitando os pedestres e outros veículos.

Reconhecimento de voz: Um viajante telefonando para a United Airlines para reservar um voo pode ter toda a conversa guiada por um sistema automático de reconhecimento de voz e de gestão de diálogo.

Planejamento autônomo e escalonamento: A uma centena de milhões de quilômetros da Terra, o programa Remote Agent da Nasa se tornou o primeiro programa de planejamento autônomo de bordo a controlar o escalonamento de operações de uma nave espacial (Jonsson *et al*., 2000). O Remote Agent gerou planos de metas de alto nível especificadas a partir do solo e monitorou a execução daqueles planos — efetuando a detecção, o diagnóstico e a recuperação de problemas conforme eles ocorriam. O programa sucessor MAPGEN (Al-Chang *et al.,* 2004) planeja as operações diárias para a Mars Exploration Rovers da Nasa, e o MEXAR2 (Cesta *et al.,* 2007) fez o planejamento tanto logístico como científico para a missão Mars Express da Agência Espacial Europeia, em 2008.

Jogos: O DEEP BLUE da IBM se tornou o primeiro programa de computador a derrotar o campeão mundial em uma partida de xadrez, ao vencer Garry Kasparov por um placar de 3,5 a 2,5 em uma partida de exibição (Goodman e Keene, 1997). Kasparov disse que sentiu "uma nova espécie de inteligência" do outro lado do tabuleiro. A revista *Newsweek* descreveu a partida como "o último reduto do cérebro". O valor das ações da IBM teve um aumento de 18 bilhões de dólares. Campeões humanos estudaram a perda de Kasparov e foram capazes de empatar algumas partidas nos anos seguintes, mas as mais recentes partidas humano-computador foram conquistadas de maneira convincente pelo computador.

Combate a spam: A cada dia, algoritmos de aprendizagem classificam mais de um bilhão de mensagens como spam, poupando o destinatário de ter que perder tempo excluindo o que, para muitos

usuários, poderia incluir 80% ou 90% de todas as mensagens, se não fossem classificadas como spam pelos algoritmos. Devido aos spammers estarem constantemente atualizando suas táticas, é difícil que uma abordagem estática programada se mantenha, e algoritmos de aprendizagem funcionam melhor (Sahami *et al*., 1998; Goodman e Heckerman, 2004).

Planejamento logístico: Durante a crise do Golfo Pérsico em 1991, as forças armadas dos Estados Unidos distribuíram uma ferramenta denominada Dynamic Analysis and Replanning Tool, ou DART (Cross e Walker, 1994), a fim de realizar o planejamento logístico automatizado e a programação de execução do transporte. Isso envolveu até 50.000 veículos, transporte de carga aérea e pessoal ao mesmo tempo, e teve de levar em conta pontos de partida, destinos, rotas e resolução de conflitos entre todos os parâmetros. As técnicas de planejamento da IA permitiram a geração em algumas horas de um plano que exigiria semanas com outros métodos. A Defense Advanced Research Project Agency (DARPA) declarou que essa única aplicação compensou com folga os 30 anos de investimento da DARPA em IA.

Robótica: A iRobot Corporation já vendeu mais de dois milhões de aspiradores robóticos Roomba para uso doméstico. A empresa também disponibilizou o mais robusto PackBot para o Iraque e Afeganistão, onde é usado para lidar com materiais perigosos, remover explosivos e identificar a localização dos franco-atiradores.

Tradução automática: Um programa de computador traduz automaticamente do árabe para o inglês, permitindo a um nativo de língua inglesa ler o cabeçalho "Ardogan Confirma que a Turquia Não Vai Aceitar Qualquer Tipo de Pressão, Instando-os a Reconhecer Chipre". O programa utiliza um modelo estatístico construído a partir de exemplos de traduções de árabe-inglês e de exemplos de textos em inglês, totalizando dois trilhões de palavras (Brants *et al*., 2007). Nenhum dos cientistas da computação na equipe fala árabe, mas eles entendem as estatísticas e os algoritmos de aprendizado de máquina.

Esses são apenas alguns exemplos de sistemas de inteligência artificial que existem hoje em dia. Não é mágica ou ficção científica, mas ciência, engenharia e matemática, e este livro apresenta uma introdução a tudo isso.

1.5 RESUMO

Este capítulo define a IA e estabelece os fundamentos culturais sobre os quais ela se desenvolveu. Alguns pontos importantes são:

- Pessoas diferentes abordam a IA com objetivos diferentes em mente. Duas questões importantes são: Você se preocupa com o pensamento ou com o comportamento? Você quer modelar seres humanos ou trabalhar a partir de um padrão ideal?
- Neste livro, adotamos a visão de que a inteligência está relacionada principalmente a uma **ação racional**. No caso ideal, um **agente inteligente** adota a melhor ação possível em uma situação. Estudaremos o problema da criação de agentes que são inteligentes nesse sentido.
- Os filósofos (desde 400 a.C.) tornaram a IA concebível, considerando as ideias de que a mente

é, em alguns aspectos, semelhante a uma máquina, de que ela opera sobre o conhecimento codificado em alguma linguagem interna e que o pensamento pode ser usado para escolher as ações que deverão ser executadas.

- Os matemáticos forneceram as ferramentas para manipular declarações de certeza lógica, bem como declarações incertas e probabilísticas. Eles também definiram a base para a compreensão da computação e do raciocínio sobre algoritmos.
- Os economistas formalizaram o problema de tomar decisões que maximizam o resultado esperado para o tomador de decisões.
- Os neurocientistas descobriram alguns fatos sobre como a mente trabalha e a forma como ela se assemelha e se diferencia dos computadores.
- Os psicólogos adotaram a ideia de que os seres humanos e os animais podem ser considerados máquinas de processamento de informações. Os linguistas mostraram que o uso da linguagem se ajusta a esse modelo.
- Os engenheiros de computação forneceram máquinas cada vez mais poderosas que tornam possíveis as aplicações de IA.
- A teoria de controle lida com o projeto de dispositivos que agem de forma ótima com base no *feedback* do ambiente. Inicialmente, as ferramentas matemáticas da teoria de controle eram bem diferentes da IA, mas os campos estão se tornando mais próximos.
- A história da IA teve ciclos de sucesso, otimismo impróprio e quedas resultantes no entusiasmo e na subvenção. Também houve ciclos de introdução de novas abordagens criativas e de aprimoramento sistemático das melhores estratégias.
- A IA avançou mais rapidamente na última década, devido ao uso mais intenso do método científico nas experiências e na comparação entre as abordagens.
- O progresso recente na compreensão da base teórica da inteligência caminha lado a lado com os avanços na capacidade de sistemas reais. Os subcampos da IA se tornaram mais integrados, e a IA encontrou uma área de concordância com outras disciplinas.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O *status* metodológico da inteligência artificial é investigado em *The Sciences of the Artificial*, de Herb Simon (1981), que descreve áreas de pesquisas relacionadas a artefatos complexos. Ele explica como a IA pode ser visualizada ao mesmo tempo como ciência e matemática. Cohen (1995) apresenta uma visão geral da metodologia experimental dentro da IA.

O teste de Turing (Turing, 1950) foi discutido por Shieber (1994), que criticou severamente a utilidade de sua instanciação na competição Loebner Prize, e por Ford e Hayes (1995), que argumentaram que o teste em si não é útil para IA. Bringsjord (2008) deu conselhos para um juiz do teste de Turing. Shieber (2004) e Epstein *et al.* (2008) coletaram uma série de experimentos sobre o teste de Turing. *Artificial Intelligence*: *The Very Idea* de John Haugeland (1985) expõe de forma lúcida os problemas filosóficos e práticos da IA. Trabalhos anteriores significativos de IA estão compilados nas coleções de Webber e Nilsson (1981) e de Luger (1995). A *Encyclopedia of AI* (Shapiro, 1992) contém artigos de pesquisa sobre quase todos os tópicos relacionados à IA assim como a Wikipédia. Em geral, esses artigos fornecem um bom ponto de partida para o estudo da

literatura de pesquisa sobre cada tópico. Uma criteriosa e abrangente história da IA é fornecida por Nils Nillson (2009), um dos primeiros pioneiros nesse campo.

O trabalho mais recente aparece nos anais das conferências sobre IA mais importantes: a bienal International Joint Conference on AI (IJCAI), a bienal European Conference on AI (ECAI) e a National Conference on AI, conhecida principalmente como AAAI, que representa a organização que a patrocina. Os principais periódicos referentes à IA em geral são *Artificial Intelligence, Computational Intelligence*, o *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Intelligent Systems* e a revista eletrônica *Journal of Artificial Intelligence Research*. Também existem muitas conferências e periódicos dedicados a áreas específicas, que abordaremos nos capítulos apropriados. As principais associações profissionais para a IA são a American Association for Artificial Intelligence (AAAI), o ACM Special Interest Group in Artificial Intelligence (SIGART) e a Society for Artificial Intelligence and Simulation of Behaviour (AISB). A revista da AAAI *AI Magazine* contém muitos artigos sobre tópicos variados e tutoriais, e seu site, aaai.org, contém notícias, tutoriais e informações básicas.

EXERCÍCIOS

Estes exercícios foram planejados para estimular a discussão, e alguns poderiam ser definidos como projetos semestrais. Como outra alternativa, podem ser feitas tentativas preliminares para resolvê-los agora, e essas tentativas podem ser revistas após a conclusão da leitura.

1.1 Defina com suas próprias palavras: (a) inteligência, (b) inteligência artificial, (c) agente, (d) racionalidade, (e) raciocínio lógico.

1.2 Leia o artigo original de Turing sobre IA (Turing, 1950). No artigo ele discute diversas objeções sobre sua iniciativa proposta e seu teste de inteligência. Que objeções ainda exercem influência? Suas refutações ainda são válidas? Você consegue imaginar o surgimento de novas objeções de desenvolvimento desde que ele escreveu seu artigo? No artigo ele prediz que por volta do ano 2000, um computador terá 30% de probabilidade de passar em um teste de Turing de 5 minutos com um interrogador não especializado. Que chance você acha que um computador teria hoje? E daqui a 50 anos?

1.3 As ações reflexas (como recuar de um fogão quente) são racionais? São inteligentes?

1.4 Suponha que estendamos o programa ANALOGY de Evans para que possa alcançar 200 em um teste de QI. Dessa forma teríamos um programa mais inteligente que um ser humano? Explique.

1.5 A estrutura neural da lesma do mar *Aplysia* foi amplamente estudada (primeiro por Eric Kandel, Prêmio Nobel) porque tem apenas cerca de 20 mil neurônios, a maioria deles grandes e facilmente manipuláveis. Assumindo que o ciclo de tempo para um neurônio da *Aplysia* é praticamente o mesmo de um neurônio humano, como é que a capacidade de processsamento, em termos de atualizações por segundo da memória, compara-se ao computador de alta capacidade descrito na Figura 1.3?

1.6 Como a introspecção — o exame que alguém faz de seus próprios pensamentos mais íntimos — poderia ser imprecisa? Eu poderia estar errado sobre aquilo em que estou pensando? Discuta.

1.7 Até que ponto os sistemas seguintes são instâncias de inteligência artificial?

- Leitores de código de barra de supermercados.
- Menus de voz de telefones.
- Mecanismos de busca na Web.
- Algoritmos de roteamento da Internet que respondem dinamicamente ao estado da rede.

1.8 Muitos dos modelos computacionais de atividades cognitivas que têm sido propostos envolvem operações matemáticas bastante complexas, como a convolução de uma imagem com o filtro de Gauss ou encontrar o mínimo da função de entropia. A maioria dos humanos (e, certamente, todos os animais) nunca aprende esse tipo de matemática e quase ninguém consegue calcular a convolução de uma função de Gauss de cabeça. Que sentido há em dizer que o "sistema de visão" está resolvendo esse tipo de matemática enquanto a pessoa real não tem ideia de como fazê-lo?

1.9 Por que a evolução tenderia a resultar em sistemas que agem racionalmente? Quais são os objetivos de projeto de tais sistemas?

1.10 A IA é uma ciência ou engenharia? Nenhum dos dois ou ambos? Explique.

1.11 "Sem dúvida, os computadores não podem ser inteligentes — eles só podem fazer o que seus programadores determinam." Esta última afirmação é verdadeira e implica a primeira?

1.12 "Sem dúvida, os animais não podem ser inteligentes — eles só podem fazer o que seus genes determinam." Esta última afirmação é verdadeira e implica a primeira?

1.13 "Sem dúvida, animais, seres humanos e computadores não podem ser inteligentes — eles só podem fazer o que seus átomos constituintes determinam, de acordo com as leis da física." Esta última afirmação é verdadeira e implica a primeira?

1.14 Examine a literatura de IA para descobrir se as seguintes tarefas podem realmente ser resolvidas por computadores:

a. Jogar um jogo decente de tênis de mesa (pingue-pongue).

b. Dirigir no centro do Cairo, Egito.

c. Dirigir em Victorville, Califórnia.

d. Comprar mantimentos para uma semana no mercado.

e. Comprar uma semana de mantimentos na Web.

f. Jogar um jogo decente de *bridge* em nível competitivo.

g. Descobrir e provar novos teoremas matemáticos.

h. Escrever uma história intencionalmente engraçada.

i. Dar assessoria jurídica competente em uma área especializada de direito.

j. Traduzir inglês falado em sueco falado, em tempo real.

k. Executar uma operação cirúrgica complexa.

Para as tarefas hoje inviáveis, tentar descobrir quais são as dificuldades e prever quando e se alguma vez serão superadas.

1.15 Vários subcampos da IA realizaram concursos através da definição de uma tarefa-padrão, convidando os pesquisadores a dar o melhor de si. Os exemplos incluem o DARPA Grand Challenge, para carros robóticos, The International Planning Competition, o futebol robótico Robocup, o evento de recuperação de informação TREC e concursos de tradução automática, reconhecimento de voz. Investigue cinco desses concursos e descreva os progressos realizados ao longo dos anos. Até que ponto os concursos avançaram o estado da arte em IA? Até que ponto causaram prejuízo ao campo, retirando energia de novas ideias?

---

[1] Ao fazermos distinção entre comportamento *humano* e *racional*, não estamos sugerindo que os seres humanos sejam necessariamente "irracionais" no sentido de "emocionalmente instáveis" ou "insanos". Simplesmente precisamos observar que não somos perfeitos: nem todos os jogadores de xadrez são grandes mestres e, infelizmente, nem todos os seres humanos conseguem conceito A nos exames. Alguns erros sistemáticos do raciocínio humano estão catalogados em Kahneman *et al.* (1982).

[2] O *Novum Organum* é uma atualização do *Organon* de Aristóteles, ou instrumento de pensamento. Então, podemos considerar Aristóteles tanto empirista quanto racionalista.

[3] Nesse quadro, todas as declarações que fazem sentido podem ser confirmadas ou definidas como falsas por experimentação ou pela análise do significado das palavras. Por eliminar a maior parte da metafísica, como era sua intenção, o positivismo lógico era impopular em alguns círculos.

[4] Tradução direta do grego Pietro Nasseti, Editora Martin Claret, p. 63.

[5] A notação proposta por Frege para a lógica de primeira ordem — uma combinação enigmática de aspectos textuais e geométricos — nunca se tornou popular.

[6] Desde então, foi descoberto que o musaranho (*Scandentia*) tem alta proporção de cérebro em relação à massa corporal.

[7] Muitos citam Alexander Hood (1824) como possível fonte anterior.

[8] Golgi persistiu em sua convicção de que as funções do cérebro eram executadas principalmente em um meio contínuo no qual os neurônios estavam incorporados, enquanto Cajal propunha a "doutrina neuronal". Os dois compartilharam o Prêmio Nobel em 1906, mas pronunciaram discursos mutuamente antagônicos ao aceitarem o mesmo.

[9] Heath Robinson foi um cartunista famoso por suas representações de aparelhos extravagantes e absurdamente complicados para realizar tarefas diárias como passar manteiga em torradas.

[10] No período do pós-guerra, Turing queria usar esses computadores em pesquisas de IA — por exemplo, um dos primeiros programas de xadrez (Turing *et al.*, 1953). Seus esforços foram bloqueados pelo governo britânico.

[11] Esse foi o primeiro uso oficial do termo de McCarthy, *inteligência artificial.* Talvez "racionalidade computacional" tivesse sido mais preciso e menos ameaçador, mas "IA" pegou. No 50º aniversário da conferência de Dartmouth, McCarthy declarou que resistiu aos termos "computador" ou "computacional" em deferência a Norbert Weiner, que estava promovendo dispositivos cibernéticos analógicos em vez de computadores digitais.

[12] Agora Carnegie Mellon University (CMU).

[13] Newell e Simon também criaram uma linguagem de processamento de listas, a IPL, para escrever o LT. Eles não tinham nenhum compilador e fizeram a conversão para código de máquina à mão. Para evitar erros, trabalharam em paralelo, gritando números binários um para o outro à medida que escreviam cada instrução, a fim de ter certeza de que os números concordavam.

[14] Em inglês: "the spirit is willing, but the flesh is weak".

[15] Para evitar embaraços, foi criado um novo campo chamado IKBS (Intelligent Knowledge-Based Systems), porque a IA havia sido oficialmente cancelada.

[16] Alguns caracterizaram essa mudança como uma vitória dos **puros** — aqueles que pensam que as teorias da IA devem se fundamentar no rigor matemático — sobre os **impuros** — aqueles que preferem experimentar muitas ideias, escrever alguns programas e depois avaliar o que parece estar funcionando. As duas abordagens são importantes. Um deslocamento em direção à pureza implica

que o campo alcançou um nível de estabilidade e maturidade. Se essa estabilidade será interrompida por uma nova ideia impura é outra questão.

CAPÍTULO 2

Agentes inteligentes

Em que discutimos a natureza dos agentes, perfeitos ou não, a diversidade de ambientes e a consequente variedade de tipos de agentes.

O Capítulo 1 identificou o conceito de **agentes racionais** como questão central para nossa abordagem da inteligência artificial. Neste capítulo, tornaremos essa noção mais concreta. Veremos que o conceito de racionalidade pode ser aplicado a uma ampla variedade de agentes que operam em qualquer ambiente imaginável. Nosso plano neste livro é usar esse conceito para desenvolver um pequeno conjunto de princípios de projeto com a finalidade de construir sistemas de agentes bem-sucedidos — sistemas que possam ser adequadamente chamados **inteligentes**.

Começaremos examinando agentes, ambientes e o acoplamento entre eles. A observação de que alguns agentes se comportam melhor que outros leva naturalmente à ideia de agente racional — um agente que se comporta tão bem quanto possível. A medida da qualidade do comportamento de um agente depende da natureza do ambiente; alguns ambientes são mais difíceis que outros. Apresentaremos uma divisão geral dos ambientes em categorias e mostraremos como as propriedades de um ambiente influenciam o projeto de agentes adequados para esse ambiente. Descreveremos vários "esqueletos" básicos de projetos de agentes que serão utilizados no restante do livro.

2.1 AGENTES E AMBIENTES

Um **agente** é tudo o que pode ser considerado capaz de perceber seu **ambiente** por meio de **sensores** e de agir sobre esse ambiente por intermédio de **atuadores**. Essa ideia simples é ilustrada na Figura 2.1. Um agente humano tem olhos, ouvidos e outros órgãos como sensores, e tem mãos, pernas, boca e outras partes do corpo que servem como atuadores. Um agente robótico pode ter câmeras e detectores da faixa de infravermelho funcionando como sensores e vários motores como atuadores. Um agente de software recebe sequências de teclas digitadas, conteúdo de arquivos e pacotes de rede como entradas sensórias e atua sobre o ambiente exibindo algo na tela, escrevendo em arquivos e enviando pacotes de rede.

 Usamos o termo **percepção** para fazer referência às entradas perceptivas do agente em um dado instante. A **sequência de percepções** do agente é a história completa de tudo o que o agente já

percebeu. Em geral, *a escolha de ação de um agente em qualquer instante dado pode depender da sequência inteira de percepções recebidas até o momento*, mas não de percepções não recebidas. Se pudermos especificar a escolha de ação do agente para toda sequência de percepções possível, teremos dito quase tudo o que existe a dizer sobre o agente. Em termos matemáticos, afirmamos que o comportamento do agente é descrito pela **função do agente** que mapeia qualquer sequência de percepções específica para uma ação.

Podemos imaginar a *tabulação* da função do agente que descreve qualquer agente dado; para a maioria dos agentes, o resultado seria uma tabela muito grande — na verdade infinita, a menos que seja definido um limite sobre o comprimento das sequências de percepções que queremos considerar. Dado um agente para a realização de experimentos, podemos, em princípio, construir essa tabela tentando todas as sequências de percepções e registrando as ações que o agente executa em resposta.[1] É claro que a tabela é uma caracterização *externa* do agente. *Internamente*, a função do agente para um agente artificial será implementada pelo **programa do agente**. É importante manter essas duas ideias distintas. A função de agente é uma descrição matemática abstrata; o programa do agente é uma implementação concreta, executada em um sistema físico.

Para ilustrar essas ideias, usaremos um exemplo muito simples — o mundo de aspirador de pó ilustrado na Figura 2.2. Esse mundo é tão simples que podemos descrever tudo o que acontece; ele também é um mundo inventado e, portanto, podemos criar muitas variações. Esse mundo particular tem apenas dois locais: os quadrados A e B. O agente aspirador de pó percebe em que quadrado está e se existe sujeira no quadrado. Ele pode optar por mover-se para a esquerda, mover-se para a direita, aspirar a sujeira ou não fazer nada. Uma função do agente muito simples é: se o quadrado atual estiver sujo, então aspirar, caso contrário mover-se para o outro quadrado. Uma tabulação parcial da função desse agente é mostrada na Figura 2.3 e um programa do agente que o implementa aparece na Figura 2.8, página 43.

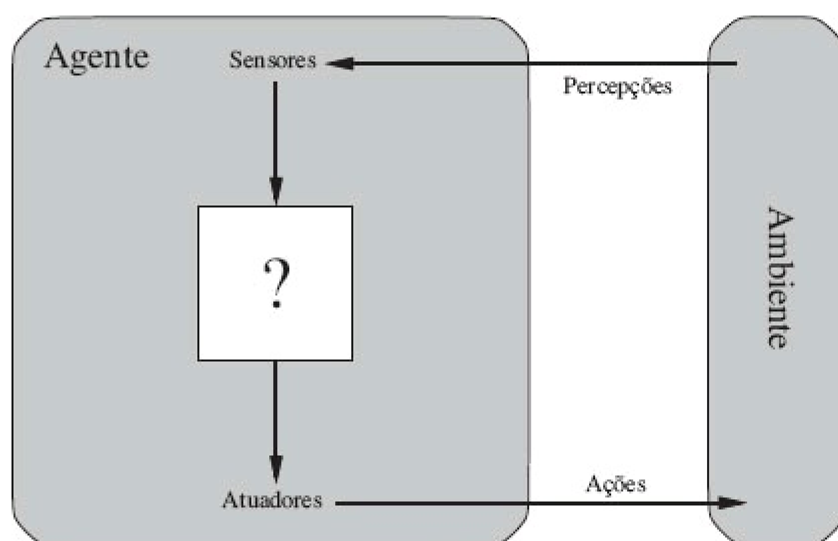


Figura 2.1 Agentes interagem com ambientes por meio de sensores e atuadores.

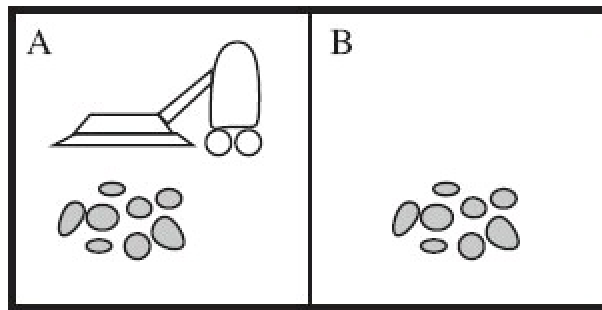


Figura 2.2 Um mundo de aspirador de pó com apenas dois locais.

Sequência de percepções	**Ação**
[*A*, *Limpo*]	*Direita*
[*A*, *Sujo*]	*Aspirar*
[*B*, *Limpo*]	*Esquerda*
[*B*, *Sujo*]	*Aspirar*
[*A*, *Limpo*], [*A*, *Limpo*]	*Direita*
[*A*, *Limpo*], [*A*, *Sujo*]	*Aspirar*
.	
.	
.	
[*A*, *Limpo*], [*A*, *Limpo*], [*A*, *Limpo*]	*Direita*
[*A*, *Limpo*], [*A*, *Limpo*], [*A*, *Sujo*]	*Aspirar*
.	
.	
.	

Figura 2.3 Tabulação parcial de uma função de agente simples correspondente ao mundo de aspirador de pó mostrado na Figura 2.2.

Examinando a Figura 2.3, vemos diversos agentes do mundo de aspirador de pó que podem ser definidos simplesmente preenchendo-se de várias maneiras a coluna da direita. Então, a pergunta óbvia é: *Qual é a maneira correta de preencher a tabela?* Em outras palavras, o que torna um agente bom ou ruim, inteligente ou estúpido? Responderemos a essas perguntas na próxima seção.

Antes de fecharmos esta seção, enfatizaremos que a noção de um agente deve ser vista como uma ferramenta para analisar sistemas, não como uma caracterização absoluta que divide o mundo em agentes e não agentes. Poderíamos visualizar uma calculadora portátil como um agente que escolhe a ação de exibir "4" ao receber a sequência de percepções "2 + 2 = ", mas tal análise dificilmente ajudaria nossa compreensão da calculadora. De certo modo, todas as áreas de engenharia podem ser vistas como projetar artefatos que interagem com o mundo; a IA opera no que os autores consideram ser o final mais interessante do espectro, onde os artefatos têm consideráveis recursos computacionais e o ambiente de tarefa requer uma tomada de decisão não trivial.

2.2 BOM COMPORTAMENTO: O CONCEITO DE RACIONALIDADE

Um **agente racional** é aquele que faz tudo certo — em termos conceituais, toda entrada na tabela correspondente à função do agente é preenchida de forma correta. É óbvio que fazer tudo certo é melhor do que fazer tudo errado; porém, o que significa fazer tudo certo?

Responderemos a essa antiga questão de uma forma antiquada: considerando as *consequências* do comportamento do agente. Quando um agente é colocado em um ambiente, gera uma sequência de ações de acordo com as percepções que recebe. Essa sequência de ações faz com que o ambiente passe por uma sequência de estados. Se a sequência for desejável, o agente teve bom desempenho. Essa noção de "desejável" é capturada por uma **medida de desempenho** que avalia qualquer sequência dada dos estados do ambiente.

Observe que dissemos estados do *ambiente*, não estados do *agente*. Se definirmos sucesso em termos da opinião do agente do seu próprio desempenho, um agente poderia alcançar a racionalidade perfeita simplesmente iludindo-se de que seu desempenho foi perfeito. Os agentes humanos em particular são notórios por ficar com "dor de cotovelo", acreditando que realmente não queriam alguma coisa (por exemplo, um Prêmio Nobel) depois de não conseguir.

Obviamente, não há uma medida de desempenho fixa para todas as tarefas e agentes; normalmente, um projetista vai desenvolver uma adequada às circunstâncias. Não é tão fácil como parece. Considere, por exemplo, o agente aspirador de pó da seção anterior. Poderíamos propor medir o desempenho pela quantidade de sujeira aspirada em um único turno de oito horas. É claro que, no caso de um agente racional, você obtém aquilo que solicita. Um agente racional pode maximizar essa medida de desempenho limpando a sujeira e, em seguida, despejando-a toda no chão, depois limpando novamente, e assim por diante. Uma medida de desempenho mais apropriada recompensaria o agente por deixar o chão limpo. Por exemplo, ele poderia ser recompensado por cada quadrado limpo em cada período (talvez com uma penalidade pela eletricidade consumida e pelo ruído gerado). *Como regra geral*, *é melhor projetar medidas de desempenho de acordo com o resultado realmente desejado no ambiente*, *em vez de criá-las de acordo com o comportamento esperado do agente*.

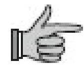
Mesmo que as armadilhas óbvias sejam evitadas, ainda existem algumas questões complexas para desembaraçar. Por exemplo, a noção de "chão limpo" no parágrafo anterior se baseia na limpeza média ao longo do tempo. Ainda assim, a mesma limpeza média pode ser alcançada por dois agentes diferentes, um dos quais faz o trabalho tedioso de limpeza o tempo todo, enquanto o outro limpa energicamente, mas faz longas pausas. A estratégia preferível pode parecer um detalhe secundário da ciência do trabalho doméstico, mas de fato é uma profunda questão filosófica com extensas implicações. O que é melhor: uma vida aventureira, cheia de altos e baixos, ou uma existência segura, porém monótona? O que é melhor: uma economia em que todos vivam em pobreza moderada ou aquela em que alguns vivem em plena riqueza enquanto outros são muito pobres? Deixaremos essas perguntas como exercício para o leitor.

2.2.1 Racionalidade

A definição do que é racional em qualquer instante dado depende de quatro fatores:

- A medida de desempenho que define o critério de sucesso.
- O conhecimento prévio que o agente tem do ambiente.
- As ações que o agente pode executar.
- A sequência de percepções do agente até o momento.

Isso conduz a uma **definição de um agente racional**:

 *Para cada sequência de percepções possível, um agente racional deve selecionar uma ação que se espera venha a maximizar sua medida de desempenho, dada a evidência fornecida pela sequência de percepções e por qualquer conhecimento interno do agente.*

Considere o agente aspirador de pó simples que limpa um quadrado se ele estiver sujo e passa para o outro quadrado se o primeiro não estiver sujo; essa é a função do agente tabulada na Figura 2.3. Esse é um agente racional? Depende! Primeiro, precisamos dizer o que é a medida de desempenho, o que se conhece sobre o ambiente e quais são os sensores e atuadores que o agente tem. Vamos supor que:

- A medida de desempenho ofereça o prêmio de um ponto para cada quadrado limpo em cada período de tempo, ao longo de um "tempo de vida" de 1.000 passos de tempo.
- A "geografia" do ambiente seja conhecida *a priori* (Figura 2.2), mas a distribuição da sujeira e a posição inicial do agente não sejam previamente conhecidas. Quadrados limpos permanecem limpos, e a aspiração limpa o quadrado atual. As ações *Esquerda* e *Direita* movem o agente para a esquerda e para a direita, exceto quando isso leva o agente para fora do ambiente; nesse caso, o agente permanece onde está.
- As únicas ações disponíveis são *Esquerda*, *Direita* e *Aspirar*.
- O agente percebe corretamente sua posição e se essa posição contém sujeira.

Afirmamos que, *sob essas circunstâncias*, o agente é de fato racional; espera-se que seu desempenho seja pelo menos tão alto quanto o de qualquer outro agente. O Exercício 2.2 lhe pede para provar esse fato.

Podemos ver facilmente que o mesmo agente seria irracional sob circunstâncias diferentes. Por exemplo, uma vez que toda a sujeira seja limpa, o agente oscila desnecessariamente de um lado para outro; se a medida de desempenho incluir uma penalidade de um ponto para cada movimento à esquerda ou à direita, o agente ficará em má situação. Um agente melhor para esse caso não faria nada se tivesse certeza de que todos os quadrados estão limpos. Se quadrados limpos puderem ficar sujos novamente, o agente deve ocasionalmente verificar e voltar a limpá-los, se necessário. Se a geografia do ambiente for desconhecida, o agente precisará explorá-la, em vez de se fixar nos quadrados *A* e *B*. O Exercício 2.2 pede para projetar agentes para esses casos.

2.2.2 Onisciência, aprendizado e autonomia

Precisamos ter o cuidado de distinguir entre racionalidade e **onisciência**. Um agente onisciente

sabe o resultado *real* de suas ações e pode agir de acordo com ele; porém, a onisciência é impossível na realidade. Considere o exemplo a seguir: estou caminhando nos Champs Elysées e de repente vejo um velho amigo do outro lado da rua. Não existe nenhum tráfego perto e não tenho nenhum outro compromisso; assim, sendo racional, começo a atravessar a rua. Enquanto isso, a 10.000 metros de altura, a porta do compartimento de carga se solta de um avião[2] e, antes de chegar ao outro lado da rua, sou atingido. Foi irracional atravessar a rua? É improvável que a notícia de minha morte fosse "idiota tenta cruzar rua".

Esse exemplo mostra que racionalidade não é o mesmo que perfeição. A racionalidade maximiza o desempenho *esperado*, enquanto a perfeição maximiza o desempenho *real*. Fugir à exigência de perfeição não é apenas uma questão de ser justo com os agentes. Se esperarmos que um agente realize aquela que virá a ser a melhor ação após o fato, será impossível projetar um agente para satisfazer essa especificação, a menos que melhoremos o desempenho de bolas de cristal ou máquinas do tempo.

Portanto, nossa definição de racionalidade não exige onisciência porque a escolha racional só depende da sequência de percepções *até o momento*. Também devemos assegurar que não permitimos que o agente se engaje sem querer em atividades decididamente pouco inteligentes. Por exemplo, se um agente não olhar para os dois lados antes de atravessar uma estrada movimentada, sua sequência de percepções não o informará de que existe um grande caminhão se aproximando em alta velocidade. Nossa definição de racionalidade afirmaria que agora é correto atravessar a estrada? Longe disso! Primeiro, não seria racional atravessar a estrada dada essa sequência de percepções pouco informativa: o risco de acidente resultante de atravessar a estrada sem olhar para os lados é muito grande. Em segundo lugar, um agente racional deveria escolher a ação "olhar" antes de iniciar a travessia porque olhar ajuda a maximizar o desempenho esperado. A realização de ações *com a finalidade de modificar percepções futuras* — às vezes chamada **coleta de informações** — é uma parte importante da racionalidade e é abordada em profundidade no Capítulo 16. Um segundo exemplo de coleta de informações é dado pela **exploração** que tem de ser empreendida por um agente aspirador de pó em um ambiente inicialmente desconhecido.

Nossa definição exige um agente racional não apenas para coletar informações, mas também para **aprender** tanto quanto possível a partir do que ele percebe. A configuração inicial do agente poderia refletir algum conhecimento prévio do ambiente, mas, à medida que o agente ganha experiência, isso pode ser modificado e ampliado. Existem casos extremos em que o ambiente é completamente conhecido *a priori*. Em tais casos, o agente não precisa perceber ou aprender; ele simplesmente age de forma correta. É claro que tais agentes são muito frágeis. Considere o humilde besouro de esterco. Depois de cavar seu ninho e depositar os ovos, ele busca uma bola de esterco em um monte próximo para fechar a entrada. Se, *durante o percurso*, a bola de esterco for removida de suas garras, o besouro seguirá em frente e imitará o fechamento do ninho com a bola de esterco inexistente, sem notar que ela foi retirada. A evolução construiu uma suposição sobre o comportamento do besouro e, quando essa hipótese é violada, resulta um comportamento malsucedido. A vespa Sphex é um pouco mais inteligente. A fêmea da Sphex cava uma cova, sai, pica uma lagarta e a arrasta até a borda da cova, entra novamente na cova para verificar se tudo está bem, arrasta a lagarta para dentro e deposita seus ovos. A lagarta servirá como alimento quando os ovos eclodirem. Até aqui tudo bem, mas se um entomologista afastar a lagarta algumas polegadas enquanto a fêmea estiver fazendo a

verificação, ela voltará à etapa de "arrastar" de seu plano e continuará o plano sem modificação, mesmo depois de dezenas de intervenções de afastamento de lagartas. A Sphex é incapaz de aprender que seu plano inato está falhando e, portanto, não o modificará.

Quando um agente se baseia no conhecimento anterior de seu projetista e não em suas próprias percepções, dizemos que o agente não tem **autonomia**. Um agente racional deve ser autônomo — ele deve aprender o que puder para compensar um conhecimento prévio parcial ou incorreto. Por exemplo, um agente aspirador de pó que aprende a prever onde e quando aparecerá mais sujeira funcionará melhor que um agente incapaz de fazer essa previsão. Na prática, raramente se exige autonomia completa desde o início: quando o agente tem pouca ou nenhuma experiência, ele deve agir ao acaso, a menos que o projetista tenha dado a ele alguma assistência. Então, da mesma forma que a evolução fornece aos animais reflexos internos suficientes para que eles possam sobreviver pelo tempo necessário para aprenderem por si mesmos, seria razoável fornecer a um agente de inteligência artificial algum conhecimento inicial, bem como habilidade para aprender.

Depois de adquirir experiência suficiente sobre seu ambiente, o comportamento de um agente racional pode se tornar efetivamente *independente* de seu conhecimento anterior. Em consequência disso, a incorporação do aprendizado permite projetar um único agente racional que terá sucesso em ampla variedade de ambientes.

2.3 A NATUREZA DOS AMBIENTES

Agora que temos uma definição de racionalidade, estamos quase prontos para pensar em construir agentes racionais. Porém, primeiro devemos pensar em **ambientes de tarefas**, que são essencialmente os "problemas" para os quais os agentes racionais são as "soluções". Começamos mostrando como especificar um ambiente de tarefa ilustrando o processo com vários exemplos. Em seguida, mostramos que há vários tipos de ambientes de tarefas. O tipo de ambiente de tarefa afeta diretamente o projeto apropriado para o programa do agente.

2.3.1 Especificando o ambiente de tarefa

Em nossa discussão sobre a racionalidade do agente aspirador de pó simples, tivemos de especificar a medida de desempenho, o ambiente e os atuadores e sensores do agente. Agruparemos todos esses itens sob o título **ambiente da tarefa**. Para os leitores que gostam de acrônimos, chamaremos essa descrição de **PEAS** (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors — desempenho, ambiente, atuadores, sensores). Ao projetar um agente, a primeira etapa deve ser sempre especificar o ambiente de tarefa de forma tão completa quanto possível.

O mundo do aspirador de pó foi um exemplo simples; vamos considerar um problema mais complexo: um motorista de táxi automatizado. Utilizaremos esse exemplo em todo o restante do capítulo. Devemos destacar, antes que o leitor fique alarmado, que um táxi totalmente automatizado no momento está um pouco além da capacidade da tecnologia atual (veja, na página 28, uma descrição de um robô motorista). A tarefa completa de dirigir é extremamente *aberta*. Não existe

nenhum limite para as novas combinações de circunstâncias que podem surgir — outra razão para termos escolhido essa tarefa como foco de discussão. A Figura 2.4 resume a descrição PEAS para o ambiente de tarefa do táxi. Descreveremos cada elemento com mais detalhes nos próximos parágrafos.

Tipo de agente	Medida de desempenho	Ambiente	Atuadores	Sensores
Motorista de táxi	Viagem segura, rápida, dentro da lei, confortável, maximizar lucros	Estradas, outros tipos de tráfego, pedestres, clientes	Direção, acelerador, freio, sinal, buzina, visor	Câmeras, sonar, velocímetro, GPS, hodômetro, acelerômetro, sensores do motor, teclado

Figura 2.4 Descrição de PEAS do ambiente de tarefa para um táxi automatizado.

Primeiro, que **medida de desempenho** gostaríamos que nosso motorista automatizado tivesse como objetivo? As qualidades desejáveis incluem chegar ao destino correto, minimizar o consumo de combustível e desgaste, minimizar o tempo e/ou o custo de viagem, minimizar as violações às leis de trânsito e as perturbações a outros motoristas, maximizar a segurança e o conforto dos passageiros e maximizar os lucros. É óbvio que alguns desses objetivos serão conflitantes; então será necessário fazer uma escolha.

Em seguida, qual é o **ambiente** de direção que o táxi enfrentará? Qualquer motorista de táxi deve lidar com diversos tipos de estradas, variando desde estradas rurais e avenidas urbanas até rodovias com 12 pistas. As estradas contêm outros tipos de tráfego, pedestres, animais perdidos, trabalhadores na pista, policiamento, poças e buracos. O táxi também deve interagir com passageiros potenciais e reais. Existem ainda algumas escolhas opcionais. O táxi poderia precisar operar no sul da Califórnia, onde a neve raramente é um problema, ou no Alasca, onde ela normalmente é um problema. Ele sempre poderia estar dirigindo no lado direito da pista ou talvez quiséssemos que ele fosse flexível o bastante para dirigir no lado esquerdo quando estivesse na Inglaterra ou no Japão. É óbvio que, quanto mais restrito o ambiente, mais fácil se torna o problema de projetar.

Os **atuadores** para um táxi automatizado incluem aqueles disponíveis para um motorista humano: controle sobre o motor através do acelerador e controle sobre a direção e a frenagem. Além disso, ele precisará da saída para uma tela de exibição ou um sintetizador de voz para se comunicar com os passageiros e, talvez, de algum meio para se comunicar com outros veículos, de forma educada ou não.

Os **sensores** básicos do táxi vão incluir uma ou mais câmeras de TV controláveis para que possa observar a estrada, que podem ser potencializadas com infravermelho ou sensor sonar para detectar distâncias de outros carros e obstáculos. Para evitar multas por excesso de velocidade, o táxi deverá possuir velocímetro, e, para controlar o veículo de forma correta, especialmente em curvas, deverá ter um acelerômetro. Para conhecer o estado mecânico do veículo, será necessário o conjunto habitual de sensores do motor, combustível e sistema elétrico. Como muitos motoristas humanos, pode querer um sistema de posicionamento global por satélite (GPS) para não se perder. Finalmente,

ele precisará de um teclado ou microfone para que o passageiro possa solicitar um destino.

N a Figura 2.5, esboçamos os elementos básicos do PEAS para diversos tipos de agentes. Exemplos adicionais aparecem no Exercício 2.4. Talvez seja surpresa para alguns leitores que a nossa lista de tipos de agentes inclua alguns programas que operam no ambiente completamente artificial definido pela entrada no teclado e pela saída de caracteres em uma tela. Alguém poderia dizer: "Certamente, esse não é um ambiente real, é?" De fato, o que importa não é a distinção entre ambientes "reais" e "artificiais", mas a complexidade do relacionamento entre o comportamento do agente, a sequência de percepções gerada pelo ambiente e a medida de desempenho. Alguns ambientes "reais" na realidade são bastante simples. Por exemplo, um robô projetado para inspecionar peças à medida que elas chegam em uma correia transportadora pode fazer uso de uma série de suposições simplificadoras: que a iluminação será sempre perfeita, que os únicos itens na correia transportadora serão peças de um tipo que ele conhece e que apenas duas ações serão possíveis (aceitar ou rejeitar).

Tipo de agente	**Medida de desempenho**	**Ambiente**	**Atuadores**	**Sensores**
Sistema de diagnóstico médico	Paciente saudável, minimizar custos	Paciente, hospital, equipe	Exibir perguntas, testes, diagnósticos, tratamentos, indicações	Entrada pelo teclado para sintomas, descobertas, respostas do paciente
Sistema de análise de imagens de satélite	Definição correta da categoria da imagem	Link de transmissão de satélite em órbita	Exibir a categorização da cena	Arrays de pixels em cores
Robô de seleção de peças	Porcentagem de peças em bandejas corretas	Correia transportadora com peças; bandejas	Braço e mão articulados	Câmera, sensores angulares articulados
Controlador de refinaria	Maximizar pureza, rendimento, segurança	Refinaria, operadores	Válvulas, bombas, aquecedores, mostradores	Sensores de temperatura, pressão, produtos químicos
Instrutor de inglês interativo	Maximizar nota de aluno em teste	Conjunto de alunos, ambiente de testes	Exibir exercícios, sugestões, correções	Entrada pelo teclado

Figura 2.5 Exemplos de tipos de agentes e suas descrições PEAS.

Em contraste, existem alguns **agentes de software** (ou robôs de software ou, ainda, **softbots**) em ambientes ricos e ilimitados. Imagine um softbot operador de website, projetado para vasculhar fontes de notícias da Internet e mostrar os itens interessantes a seus clientes, enquanto vende espaço

de publicidade para gerar renda. Para funcionar bem, ele precisará de algumas habilidades de processamento de linguagem natural, precisará aprender o que interessa a cada usuário e investidor e terá de mudar seus planos dinamicamente — por exemplo, quando a conexão para uma fonte de notícias cair ou quando uma nova fonte estiver on-line. A Internet é um ambiente cuja complexidade rivaliza com a do mundo físico e cujos habitantes incluem muitos agentes artificiais e humanos.

2.3.2 Propriedades de ambientes de tarefas

A variedade de ambientes de tarefas que podem surgir em IA é sem dúvida vasta. Entretanto, podemos identificar um número bastante reduzido de dimensões ao longo das quais os ambientes de tarefas podem ser divididos em categorias. Em grande parte, essas dimensões determinam o projeto apropriado de agentes e a aplicabilidade de cada uma das principais famílias de técnicas de implementação de agentes. Primeiro, listamos as dimensões, depois analisamos vários ambientes de tarefas para ilustrar as ideias. Aqui, as definições são informais; os capítulos posteriores fornecerão declarações e exemplos mais precisos de cada tipo de ambiente.

Completamente observável *versus* **parcialmente observável**: Se os sensores de um agente permitem acesso ao estado completo do ambiente em cada instante, dizemos que o ambiente de tarefa é completamente observável. Um ambiente de tarefa é de fato completamente observável se os sensores detectam todos os aspectos que são *relevantes* para a escolha da ação; por sua vez, a relevância depende da medida de desempenho. Ambientes completamente observáveis são convenientes porque o agente não precisa manter qualquer estado interno para acompanhar as mudanças do mundo. Um ambiente poderia ser parcialmente observável devido ao ruído e a sensores imprecisos ou porque partes do estado estão simplesmente ausentes nos dados do sensor — por exemplo, um agente aspirador de pó com apenas um sensor de sujeira local não pode saber se há sujeira em outros quadrados, e um táxi automatizado não pode saber o que outros motoristas estão pensando. Se o agente não tiver sensores, o ambiente será **inobservável**. Alguém poderia pensar que, nesses casos, a situação do agente fica desesperadora, mas, como discutiremos no Capítulo 4, os objetivos do agente ainda poderão ser alcançáveis, e em alguns casos, com certeza.

Agente único *versus* **multiagente:** A distinção entre ambientes de agente único e de multiagente pode parecer bastante simples. Por exemplo, um agente que resolve um jogo de palavras cruzadas sozinho está claramente em um ambiente de agente único, enquanto um agente que joga xadrez está em um ambiente de dois agentes. Porém, existem algumas questões sutis. Primeiro, descrevemos como uma entidade *pode* ser visualizada como um agente, mas não explicamos que entidades *devem* ser visualizadas como agentes. Um agente *A* (por exemplo, o motorista de táxi) tem de tratar um objeto *B* (outro veículo) como um agente ou ele pode ser tratado apenas como um objeto comportando-se de acordo com as leis da física, análogo às ondas do mar ou às folhas espalhadas pelo vento? A distinção fundamental é saber se o comportamento de *B* é ou não melhor descrito como a maximização de uma medida de desempenho cujo valor depende do comportamento do agente *A*. Por exemplo, em xadrez, a entidade oponente *B* está tentando maximizar sua medida de desempenho que, pelas regras de xadrez, minimiza a medida de desempenho do agente *A*. Desse modo, o jogo de xadrez é um ambiente de multiagente **competitivo**. Por outro lado, no ambiente de direção de um táxi,

evitar colisões maximiza a medida de desempenho de todos os agentes; assim, esse é um ambiente de multiagente parcialmente **cooperativo**. Ele também é parcialmente competitivo porque, por exemplo, apenas um carro pode ocupar um espaço no estacionamento. Os problemas de projeto de agentes que surgem em ambientes de multiagentes muitas vezes são bem diferentes dos que surgem em ambientes de um único agente; por exemplo, a **comunicação** com frequência emerge como um comportamento racional em ambientes de multiagentes; em alguns ambientes competitivos parcialmente observáveis, o **comportamento aleatório** é racional porque evita as armadilhas da previsibilidade.

Determinístico *versus* **estocástico:** Se o próximo estado do ambiente é completamente determinado pelo estado atual e pela ação executada pelo agente, dizemos que o ambiente é determinístico; caso contrário, ele é estocástico. Em princípio, um agente não precisa se preocupar com a incerteza em um ambiente completamente observável e determinístico. (Na nossa definição, ignoramos a incerteza que surge exclusivamente das ações de outros agentes em um ambiente multiagente; assim, um jogo pode ser determinístico, mesmo sendo cada agente incapaz de predizer as ações dos outros.) Porém, se o ambiente for parcialmente observável, ele poderá *parecer* estocástico. A maioria das situações reais é tão complexa que é impossível acompanhar todos os aspectos não observados; para finalidades práticas devem ser tratados como estocásticos. O motorista de táxi é claramente estocástico nesse sentido porque nunca se pode prever o comportamento do tráfego com exatidão; além disso, pode ocorrer o estouro de um pneu e a falha de um motor sem aviso prévio. O mundo do aspirador de pó que descrevemos é determinístico, mas as variações podem incluir elementos estocásticos, como o aparecimento de sujeira ao acaso e um mecanismo de sucção não confiável (Exercício 2.13). Dizemos que um ambiente é incerto se não for totalmente observável ou determinístico. Observação final: o nosso uso da palavra “estocástico” geralmente implica que a incerteza sobre os resultados é quantificada em termos de probabilidades; um ambiente **não determinístico** é aquele em que as ações são caracterizadas por seus resultados *possíveis*, sem probabilidade associada a ele. As descrições do ambiente não determinístico são normalmente associadas às medidas de desempenho que exigem que o agente tenha sucesso em *todos* os resultados *possíveis* de suas ações.

Episódico *versus* **sequencial:** Em um ambiente de tarefa episódico, a experiência do agente é dividida em episódios atômicos. Em cada episódio, o agente recebe uma percepção e em seguida executa uma única ação. É crucial que o episódio seguinte não dependa das ações executadas em episódios anteriores. Em ambientes episódicos, a escolha da ação em cada episódio só depende do próprio episódio. Muitas tarefas de classificação são episódicas. Por exemplo, um agente que tem de localizar peças defeituosas em uma linha de montagem baseia cada decisão na peça atual, independentemente das decisões anteriores; além disso, a decisão atual não afeta o fato da próxima peça estar ou não com defeito. Por outro lado, em ambientes sequenciais, a decisão atual poderia afetar todas as decisões futuras.[3] Jogar xadrez e dirigir um táxi são sequenciais: em ambos os casos, ações em curto prazo podem ter consequências a longo prazo. Ambientes episódicos são muito mais simples que ambientes sequenciais porque o agente não precisa pensar à frente.

Estático *versus* **dinâmico:** Se o ambiente puder se alterar enquanto um agente está deliberando, dizemos que o ambiente é dinâmico para esse agente; caso contrário, ele é estático. Ambientes estáticos são fáceis de manipular porque o agente não precisa continuar a observar o mundo enquanto está decidindo sobre a realização de uma ação nem precisa se preocupar com a passagem do tempo.

Por outro lado, ambientes dinâmicos estão continuamente perguntando ao agente o que ele deseja fazer; se ele ainda não tiver se decidido, isso será considerado a decisão de não fazer nada. Se o próprio ambiente não mudar com a passagem do tempo, mas o nível de desempenho do agente se alterar, diremos que o ambiente é **semidinâmico**. O ambiente em que se dirige um táxi é claramente dinâmico: os outros carros e o próprio táxi continuam a se mover enquanto o algoritmo de direção hesita sobre o que fazer em seguida. O jogo de xadrez, quando jogado com a contagem do tempo, é semidinâmico. O jogo de palavras cruzadas é estático.

Discreto *versus* **contínuo:** A distinção entre discreto e contínuo aplica-se ao *estado* do ambiente, ao modo como o *tempo* é tratado, e ainda às *percepções* e *ações* do agente. Por exemplo, um ambiente de jogo de xadrez tem um número finito de estados distintos (excluindo o relógio). O xadrez também tem um conjunto discreto de percepções e ações. Dirigir um táxi é um problema de estado contínuo e tempo contínuo: a velocidade e a posição do táxi e dos outros veículos passam por um intervalo de valores contínuos e fazem isso suavemente ao longo do tempo. As ações de dirigir um táxi também são contínuas (ângulos de rotação do volante etc.). A entrada proveniente de câmeras digitais é discreta, em termos estritos, mas em geral é tratada como a representação de intensidades e posições que variam continuamente.

Conhecido *versus* **desconhecido**: Estritamente falando, essa distinção não se refere ao ambiente em si, mas ao estado de conhecimento do agente (ou do projetista) sobre as "leis da física" no meio ambiente. Em um ambiente conhecido, são fornecidas as saídas (ou probabilidades das saídas se o ambiente for estocástico) para todas as ações. Obviamente, se o ambiente for desconhecido, o agente terá de aprender como funciona, a fim de tomar boas decisões. Observe que a distinção entre os ambientes conhecido e desconhecido não é a mesma que entre ambientes totalmente e parcialmente observáveis. É perfeitamente possível para um ambiente *conhecido* ser *parcialmente* observável — por exemplo, em jogos de cartas solitários, eu conheço as regras, mas sou incapaz de ver as cartas que ainda não foram viradas. Por outro lado, um ambiente *desconhecido* pode ser *totalmente* observável — em um novo videogame, a tela pode mostrar o estado inteiro do jogo, mas eu ainda não sei o que os botões fazem até experimentá-los.

Como se poderia esperar, o caso mais difícil é *parcialmente observável*, *multiagente, estocástico*, *sequencial*, *dinâmico*, *contínuo e desconhecido.* Dirigir um táxi é difícil em todos esses sentidos, exceto que para a maioria dos motoristas o ambiente é conhecido. Dirigir um carro alugado em um país desconhecido, com a geografia e de leis de trânsito desconhecidas, é muito mais emocionante.

A Figura 2.6 lista as propriedades de vários ambientes familiares. Observe que as respostas nem sempre são definitivas. Por exemplo, descrevemos o robô de seleção de peças como episódico porque ele normalmente considera cada peça isoladamente. Mas, se um dia houver um grande lote de peças defeituosas, o robô deverá aprender através de várias observações que a distribuição de defeitos mudou e deverá modificar o seu comportamento para as peças subsequentes. A coluna "conhecido/desconhecido" não foi incluída porque, como explicado anteriormente, ela não é estritamente uma propriedade do ambiente. Em alguns ambientes, tais como xadrez e pôquer, é muito fácil suprir o agente com pleno conhecimento das regras, mas não deixa de ser interessante considerar como um agente poderá aprender a jogar esses jogos sem tal conhecimento.

Ambiente de tarefa	Observável	Agentes	Determinístico	Episódico	Estático	Discreto
Jogo de palavras cruzadas Xadrez com um relógio	Completamente Completamente	Único Multi	Determinístico Determinístico	Sequencial Sequencial	Estático Semi	Discreto Discreto
Pôquer Gamão	Parcialmente Completamente	Multi Multi	Estocástico Estocástico	Sequencial Sequencial	Estático Estático	Discreto Discreto
Direção de táxi Diagnóstico médico	Parcialmente Parcialmente	Multi Único	Estocástico Estocástico	Sequencial Sequencial	Dinâmico Dinâmico	Contínuo Contínuo
Análise de imagens Robô de seleção de peças	Completamente Parcialmente	Único Único	Determinístico Estocástico	Episódico Episódico	Semi Dinâmico	Contínuo Contínuo
Controlador de refinaria Instrutor interativo de inglês	Parcialmente Parcialmente	Único Multi	Estocástico Estocástico	Sequencial Sequencial	Dinâmico Dinâmico	Contínuo Discreto

Figura 2.6 Exemplos de ambientes de tarefas e suas características.

Muitas das respostas na tabela dependem da forma como o ambiente de tarefa é definido. Listamos a tarefa de diagnóstico médico como uma tarefa de agente único porque o processo de doença em um paciente não poderia ser modelado de modo proveitoso como um agente; porém, um sistema de diagnóstico médico também poderia ter de lidar com pacientes obstinados e funcionários céticos e, assim, o ambiente poderia ter um aspecto multiagente. Além disso, o diagnóstico médico é episódico se a tarefa for concebida como a seleção de um diagnóstico dada uma lista de sintomas; o diagnóstico será sequencial se a tarefa puder incluir a proposição de uma série de testes, a avaliação do progresso durante o tratamento, e assim por diante. Também há muitos ambientes episódicos em níveis mais altos que as ações individuais do agente. Por exemplo, um torneio de xadrez consiste em uma sequência de jogos; cada jogo é um episódio porque (em geral) a contribuição dos movimentos em um jogo para o desempenho global do agente não é afetada pelos movimentos de seu jogo anterior. Por outro lado, a tomada de decisões em um único jogo certamente é sequencial.

O repositório de código associado a este livro (aima.cs.berkeley.edu) inclui implementações de vários ambientes, juntamente com um simulador de ambiente de uso geral que coloca um ou mais agentes em um ambiente simulado, observa seu comportamento com o passar do tempo e os avalia de acordo com determinada medida de desempenho. Com frequência, tais experimentos são executados não para um único ambiente, mas para muitos ambientes extraídos de uma **classe de ambientes**. Por

exemplo, avaliar um motorista de táxi em tráfego simulado requer a execução de muitas simulações com diferentes condições de tráfego, iluminação e condições metereológicas. Se projetássemos o agente para um único cenário, poderíamos tirar proveito de propriedades específicas do caso particular, mas não poderíamos criar um bom projeto para dirigir de maneira geral. Por essa razão, o repositório de código também inclui um **gerador de ambientes** para cada classe de ambientes que seleciona ambientes específicos (com certas variações aleatórias) nos quais seria possível executar o agente. Por exemplo, o gerador de ambientes de aspirador de pó inicializa o padrão de sujeira e a posição do agente de forma aleatória. Então, estamos interessados no desempenho médio do agente sobre a classe de ambientes. Um agente racional para dada classe de ambientes maximiza seu desempenho médio. Os Exercícios 2.8 a 2.13 conduzem o leitor pelo processo de desenvolver uma classe de ambientes e de avaliar diversos agentes dentro dessa classe.

2.4 A ESTRUTURA DE AGENTES

Até agora fizemos referência aos agentes descrevendo o *comportamento* — a ação executada após qualquer sequência de percepções específica. Agora, teremos de seguir em frente e descrever o funcionamento interno desses agentes. O trabalho da IA é projetar o **programa do agente** que implementa a função do agente — que mapeia percepções em ações. Supomos que esse programa será executado em algum tipo de dispositivo de computação com sensores e atuadores físicos — chamamos esse conjunto de **arquitetura**:

$$agente = arquitetura + programa.$$

É óbvio que o programa que escolhermos tem de ser apropriado para a arquitetura. Se o programa recomendar ações como *Caminhar*, é melhor que a arquitetura tenha pernas. A arquitetura pode ser apenas um PC comum ou talvez um carro robótico com diversos computadores, câmeras e outros sensores a bordo. Em geral, a arquitetura torna as percepções dos sensores disponíveis para o programa, executa o programa e fornece as escolhas de ação do programa para os atuadores à medida que elas são geradas. A maior parte deste livro trata do projeto de programas de agentes, embora os Capítulos 24 e 25 lidem diretamente com os sensores e atuadores.

2.4.1 Programas de agentes

Os programas de agentes que projetaremos neste livro têm todos a mesma estrutura básica: eles recebem a percepção atual como entrada dos sensores e devolvem uma ação para os atuadores.[4] Note a diferença entre o programa do agente, que toma a percepção atual como entrada, e a função do agente, que recebe o histórico de percepções completo. O programa do agente recebe apenas a percepção atual como entrada, uma vez que nada mais está disponível do ambiente; se as ações do agente dependem da sequência de percepções inteira, o agente terá de memorizar as percepções.

Descreveremos os programas de agentes por meio da linguagem de pseudocódigo simples definida no Apêndice B (o repositório de código on-line contém implementações em linguagens de

programação reais). Por exemplo, a Figura 2.7 mostra um programa de agente bastante trivial que acompanha a sequência de percepções e depois a utiliza para realizar a indexação em uma tabela de ações, a fim de decidir o que fazer. A tabela — cujo exemplo foi dado para o mundo do aspirador de pó na Figura 2.3 — representa explicitamente a função do agente que o programa do agente incorpora. Para construir um agente racional desse modo, devemos construir uma tabela que contenha a ação apropriada para todas as sequências de percepções possíveis.

função AGENTE-DIRIGIDO-POR-TABELA(*percepção*) **retorna** uma ação
  variáveis estáticas: *percepções*, uma sequência, inicialmente vazia
    tabela, uma tabela de ações, indexada por sequências de percepções, inicialmente completamente especificada

  anexar *percepção* ao fim de *percepções*
  ação ← ACESSAR(*percepções*, *tabela*)
  retornar *ação*

Figura 2.7 O programa AGENTE-DIRIGIDO-POR-TABELA é invocado para cada nova percepção e retorna uma ação de cada vez. Ele mantém a sequência de percepções completas na memória.

É instrutivo considerar por que a abordagem orientada a tabelas para construção de agentes está condenada ao fracasso. Seja $\mathcal{P}$ o conjunto de percepções possíveis e seja T o tempo de duração do agente (o número total de percepções que ele receberá). A tabela de pesquisa conterá $\sum_{t=1}^{T} |\mathcal{P}|^t$ entradas. Considere o táxi automatizado: a entrada visual de uma única câmera chega à velocidade de aproximadamente 27 megabytes por segundo (30 quadros por segundo, 640 × 480 pixels com 24 bits de informações de cores). Isso nos dá uma tabela de pesquisa com mais de $10^{250.000.000.000}$ entradas para uma hora de direção. Até mesmo a tabela de pesquisa para o xadrez — um minúsculo e bem-comportado fragmento do mundo real — teria pelo menos 10^{150} entradas. O tamanho assustador dessas tabelas (o número de átomos no universo observável é menor que 10^{80}) significa que (a) nenhum agente físico nesse universo terá espaço para armazenar a tabela, (b) o projetista não teria tempo para criar a tabela, (c) nenhum agente poderia sequer apreender todas as entradas de tabelas corretas a partir de sua experiência e (d) mesmo que o ambiente seja simples o bastante para gerar uma tabela de tamanho viável, o projetista ainda não terá nenhuma orientação sobre como inserir as entradas da tabela.

Apesar de tudo isso, o AGENTE-DIRIGIDO-POR-TABELA *faz* o que queremos: implementa a função de agente desejada. O desafio fundamental da IA é descobrir como escrever programas que, na medida do possível, produzam um comportamento racional a partir de um pequeno programa em vez de uma grande tabela. Temos muitos exemplos mostrando que isso pode ser feito com sucesso em outras áreas: por exemplo, as enormes tabelas de raízes quadradas usadas por engenheiros e por estudantes antes da década de 1970 foram substituídas por um programa de cinco linhas que corresponde ao método de Newton e é executado em calculadoras eletrônicas. A pergunta é: a IA pode fazer pelo comportamento inteligente em geral o que Newton fez para as raízes quadradas? Acreditamos que a resposta seja sim.

No restante desta seção, descreveremos quatro tipos básicos de programas de agentes que

incorporam os princípios subjacentes a quase todos os sistemas inteligentes:

- Agentes reativos simples.
- Agentes reativos baseados em modelo.
- Agentes baseados em objetivos.
- Agentes baseados na utilidade.

Cada tipo de programa de agente combina componentes específicos de maneiras específicas para gerar ações. A Seção 2.4.6 explica, em termos gerais, como converter todos esses agentes em *agentes de apendizagem* que podem melhorar o desempenho de seus componentes de modo a gerar melhores ações. Finalmente, a Seção 2.4.7 descreve uma variedade de maneiras de como os próprios componentes podem ser representados dentro do agente. Essa variedade proporciona um princípio organizador fundamental para o campo e para o próprio livro.

2.4.2 Agentes reativos simples

O tipo mais simples de agente é o **agente reativo simples**. Esses agentes selecionam ações com base na percepção *atual*, ignorando o restante do histórico de percepções. Por exemplo, o agente aspirador de pó cuja função do agente é tabulada na Figura 2.3 é um agente reativo simples porque sua decisão se baseia apenas na posição atual e no fato de essa posição conter ou não sujeira. Um programa para esse agente é mostrado na Figura 2.8.

Note que o programa do agente aspirador de pó na realidade é muito pequeno em comparação com a tabela correspondente. A redução mais óbvia vem de se ignorar o histórico de percepções, o que reduz o número de possibilidades de 4^T para apenas 4. Uma pequena redução adicional vem do fato de que, quando o quadrado atual está sujo, a ação não depende da posição em que o agente esteja.

Comportamentos reativos simples ocorrem mesmo em ambientes mais complexos. Imagine-se como o motorista do táxi automatizado. Se o carro da frente frear e suas luzes de freio se acenderem, você deve notar esse fato e começar a frear. Em outras palavras, algum processamento é realizado de acordo com a entrada visual para estabelecer a condição que chamamos de "O carro da frente está freando". Então, isso ativa alguma conexão estabelecida no programa do agente para a ação "começar a frear". Chamaremos tal conexão de **regra condição-ação**,[5] escrita como:

se *carro-da-frente-está-freando* **então** *começar-a-frear.*

Os seres humanos também têm muitas dessas conexões, algumas das quais são respostas aprendidas (como dirigir) e outras são reflexos inatos (como piscar quando algo se aproxima de seu olho). No decorrer do livro, veremos várias maneiras diferentes de aprender e implementar tais conexões.

O programa da Figura 2.8 é específico para um determinado ambiente do aspirador de pó. Uma abordagem mais geral e flexível consiste em primeiro construir um interpretador de uso geral para regras condição-ação e depois criar conjuntos de regras para ambientes de tarefas específicos. A Figura 2.9 fornece a estrutura desse programa geral em forma esquemática, mostrando como as regras

condição-ação permitem ao agente fazer a conexão entre percepção e ação (não se preocupe com o fato de esse assunto parecer trivial; ele ficará mais interessante em breve). Utilizamos retângulos para denotar o estado interno atual do processo de decisão do agente e elipses para representar as informações suplementares usadas no processo. O programa do agente, que também é muito simples, é mostrado na Figura 2.10. A função INTERPRETAR-ENTRADA gera uma descrição abstrata do estado atual a partir da percepção, e a função REGRA-CORRESPONDENTE retorna a primeira regra no conjunto de regras que corresponde à descrição de estado dada. Observe que a descrição em termos de "regras" e "correspondência" é puramente conceitual; as implementações reais podem ser tão simples quanto uma coleção de portas lógicas que implementam um circuito booleano.

Os agentes reativos simples têm a admirável propriedade de serem simples, mas se caracterizam por ter inteligência limitada. O agente da Figura 2.10 funcionará *somente se a decisão correta puder ser tomada com base apenas na percepção atual, ou seja, apenas se o ambiente for completamente observável*. Até mesmo uma pequena impossibilidade de observação pode causar sérias dificuldades. Por exemplo, a regra de frenagem apresentada anteriormente pressupõe que a condição *carro-da-frente-está-freando* pode ser determinada a partir da percepção atual — um único quadro de vídeo. Funciona se o carro tiver na frente uma luz de freio central. Infelizmente, modelos mais antigos têm configurações diferentes de lanternas, luzes de freio e luzes de setas, e nem sempre é possível saber por uma única imagem se o carro está freando. Um agente reativo simples que dirigisse atrás de um carro desse tipo frearia contínua e desnecessariamente ou, pior ainda, nunca frearia.

função AGENTE-ASPIRADOR-DE-PÓ-REATIVO ([*posição*, *situação*]) **retorna** uma ação*
  se *situação* = *Sujo* **então retorna** *Aspirar*
  senão se *posição* = *A* **então retorna** *Direita*
  senão se *posição* = *B* **então retorna** *Esquerda*

Figura 2.8 Programa do agente para um agente reativo simples no ambiente de aspirador de pó de dois estados. Esse programa implementa a função do agente tabulada na Figura 2.3.
* *Nota do revisor técnico:* Como aqui se trata de "pseudocódigo", é possível traduzir os comandos para facilitar a leitura. Apenas não se traduz quando se trata de uma linguagem de programação real.

Podemos ver um problema semelhante surgindo no mundo de aspirador de pó. Suponha que um agente aspirador de pó reativo simples seja destituído de seu sensor de posição e tenha apenas um sensor de sujeira. Tal agente tem apenas duas percepções possíveis: [*Sujo*] e [*Limpo*]. Ele pode *Aspirar* em resposta a [*Sujo*]; o que deve fazer em resposta a [*Limpo*]? Mover-se para a *Esquerda* falhará (sempre) se ele começar no quadrado *A*, e mover-se para a *Direita* falhará (sempre) se ele começar no quadrado *B*. Com frequência, laços de repetição infinitos são inevitáveis no caso de agentes reativos simples operando em ambientes parcialmente observáveis.

É possível escapar de laços de repetição infinitos se o agente puder tornar suas ações **aleatórias**. Por exemplo, se o agente aspirador de pó perceber [*Limpo*], ele pode jogar uma moeda para escolher entre *Esquerda* e *Direita*. É fácil mostrar que o agente alcançará o outro quadrado usando duas etapas em média. Em seguida, se esse quadrado estiver sujo, ele limpará a sujeira e a tarefa de limpeza será concluída. Consequentemente, um agente reativo simples aleatório poderia superar um

agente reativo simples determinístico.

Mencionamos na Seção 2.3 que um comportamento aleatório do tipo correto pode ser racional em alguns ambientes multiagentes. Em ambientes de um único agente, em geral a aleatoriedade *não* é racional. Ela é um artifício útil que ajuda um agente reativo simples em algumas situações, mas, na maioria dos casos, podemos fazer muito melhor com agentes determinísticos mais sofisticados.

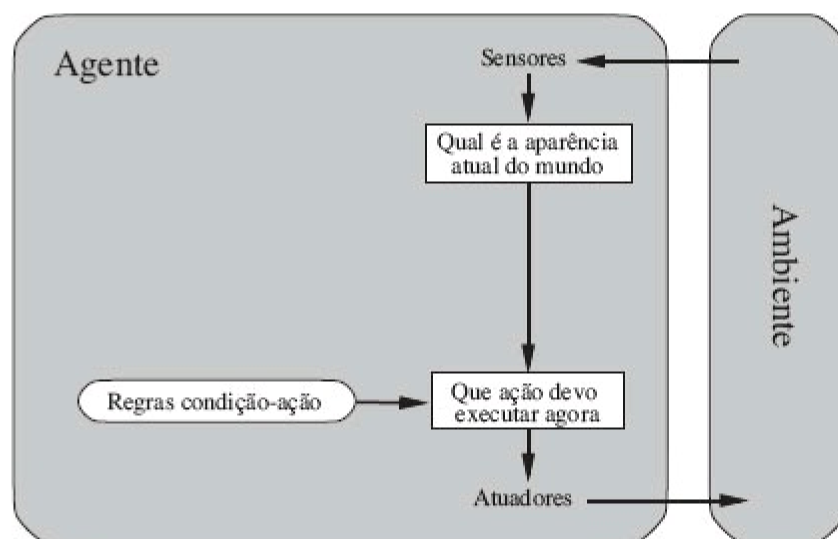


Figura 2.9 Diagrama esquemático de um agente reativo simples.

função AGENTE-REATIVO-SIMPLES (*percepção*) **retorna** uma ação
variáveis estáticas: *regras*, um conjunto de regras condição-ação

estado ← INTERPRETAR-ENTRADA (*percepção*)
regra ← REGRA-CORRESPONDENTE (*estado*, *regras*)
ação ← AÇÃO-DA-REGRA [*regra*]
retornar *ação*

Figura 2.10 Um agente reativo simples. Ele age de acordo com uma regra cuja condição corresponde ao estado atual definido pela percepção.

2.4.3 Agentes reativos baseados em modelos

O modo mais efetivo de lidar com a possibilidade de observação parcial é o agente *monitorar a parte do mundo que ele não pode ver agora*. Isto é, o agente deve manter algum tipo de **estado interno** que dependa do histórico de percepções e assim reflita pelo menos alguns dos aspectos não observados do estado atual. Para o problema do freio, o estado interno não é muito extenso — apenas o quadro anterior da câmera, que permite ao agente detectar quando duas luzes vermelhas na borda do veículo acendem ou apagam ao mesmo tempo. No caso de outras tarefas de direção, como trocar de pista, o agente precisa monitorar onde os outros carros estão, se não puder vê-los todos de uma vez. E, para que qualquer direção seja possível, o agente precisa saber onde as chaves estão.

A atualização dessas informações internas de estado à medida que o tempo passa exige que dois

tipos de conhecimento sejam codificados no programa do agente. Primeiro, precisamos de algumas informações sobre o modo como o mundo evolui independentemente do agente — por exemplo, que um carro que estiver ultrapassando em geral estará mais próximo do que estava um momento antes. Em segundo lugar, precisamos de algumas informações sobre como as ações do próprio agente afetam o mundo — de que, por exemplo, quando o agente girar o volante à direita, o carro irá virar para a direita ou de que, depois de dirigir por cinco minutos na direção norte da autoestrada, em geral ficamos cinco quilômetros ao norte de onde nos encontrávamos cinco minutos antes. Esse conhecimento de "como o mundo funciona" — seja ele implementado em circuitos booleanos simples ou em teorias científicas completas — é chamado de **modelo** do mundo. Um agente que usa tal modelo denomina-se **agente baseado em modelo**.

A Figura 2.11 fornece a estrutura do agente reativo baseado em modelo com seu estado interno, mostrando como a percepção atual é combinada com o estado interno antigo para gerar a descrição atualizada do estado atual, baseado no modelo do agente de como o mundo funciona.

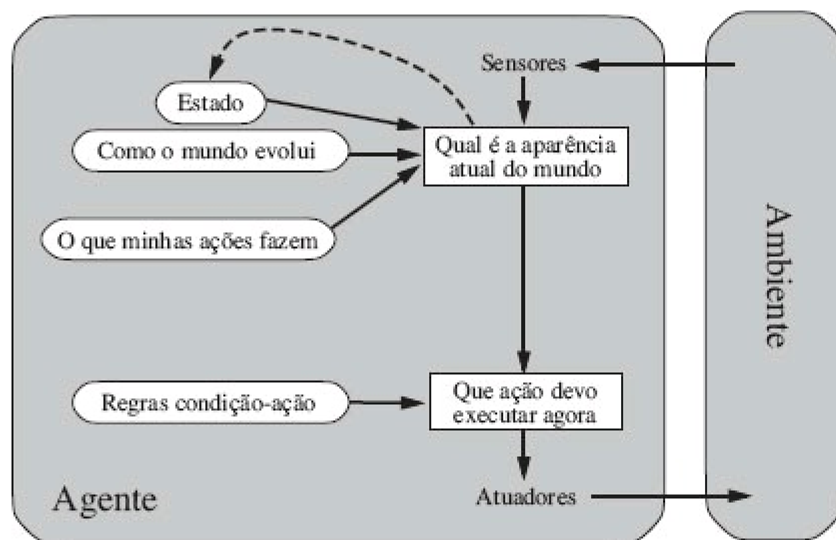


Figura 2.11 Agente reativo baseado em modelo.

O programa de agente é mostrado na Figura 2.12. A parte interessante é a função ATUALIZAR-ESTADO, responsável pela criação da descrição do novo estado interno. Os detalhes de como modelos e estados são representados variam amplamente dependendo do tipo de ambiente e da tecnologia em particular usada no projeto do agente. Nos Capítulos 4, 12, 11, 15, 17 e 25 aparecem exemplos detalhados de modelos e atualização de algoritmos.

função AGENTE-REATIVO-BASEADO-EM-MODELOS (*percepção*) **retorna** uma ação
  persistente: *estado*, a concepção do agente do estado atual do mundo
    modelo, uma descrição de como o próximo estado depende do estado atual e da ação
    regras, um conjunto de regras condição-ação
    ação, a ação mais recente, inicialmente nenhuma

  estado ← ATUALIZAR-ESTADO (*estado*, *ação*, *percepção, modelo*)
  regra ← REGRA-CORRESPONDENTE (*estado*, *regras*)

ação ← regra, AÇÃO
retornar *ação*

Figura 2.12 Um agente reativo baseado em modelo. Ele mantém o estado atual do mundo usando um modelo interno. Em seguida, ele escolhe uma ação da mesma maneira que o agente reativo simples.

Independentemente do tipo de representação utilizada, raramente é possível para o agente determinar *exatamente* o estado atual de um ambiente parcialmente observável. Em vez disso, a caixa rotulada "como o mundo se parece agora" (Figura 2.11) representa o "melhor palpite" do agente (ou, às vezes, os melhores palpites). Por exemplo, um táxi automatizado pode não ser capaz de enxergar através de um grande caminhão que parou na sua frente e talvez tenha apenas um palpite do que causou o bloqueio. Assim, a incerteza sobre o estado atual pode ser inevitável, mas o agente ainda terá que tomar uma decisão.

Um ponto talvez menos óbvio sobre o "estado" interno mantido por um agente baseado em modelo é que ele não tem que descrever "como o mundo se parece agora" em sentido literal. Por exemplo, o táxi pode estar dirigindo de volta para casa e pode ser que exista uma regra para colocar gasolina no caminho de casa se o tanque estiver pelo menos pela metade. Apesar de que "dirigir de volta para casa" *parece* um aspecto do estado do mundo, o fato do *destino* do táxi é na verdade um aspecto do estado interno do agente. Se você achar isso intrigante, considere que o táxi possa estar exatamente no mesmo lugar ao mesmo tempo, mas com a intenção de chegar a um destino diferente.

2.4.4 Agentes baseados em objetivos

Conhecer algo sobre o estado atual do ambiente nem sempre é suficiente para decidir o que fazer. Por exemplo, em um entroncamento de estradas, o táxi pode virar à esquerda, virar à direita ou seguir em frente. A decisão correta depende de onde o táxi está tentando chegar. Em outras palavras, da mesma forma que o agente precisa de uma descrição do estado atual, ele também precisa de alguma espécie de informação sobre **objetivos** que descreva situações desejáveis — por exemplo, estar no destino do passageiro. O programa de agente pode combinar isso com o modelo (as mesmas informações que foram usadas no agente reativo baseado em modelo), a fim de escolher ações que alcancem o objetivo. A Figura 2.13 mostra a estrutura do agente baseado em objetivos.

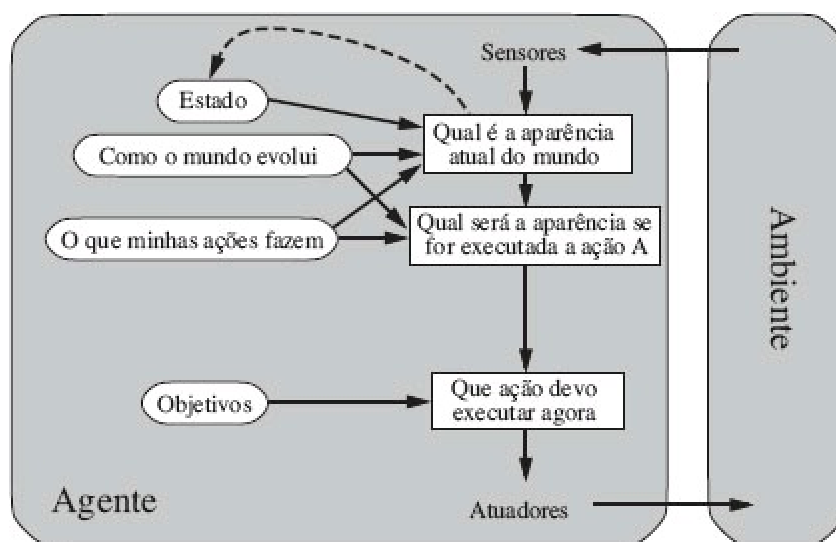


Figura 2.13 Um agente baseado em modelos e orientado pelos objetivos. Ele monitora o estado do mundo, bem como um conjunto de objetivos que está tentando atingir e escolhe uma ação que (no final) levará à realização de seus objetivos.

Às vezes, a seleção da ação baseada em objetivos é direta — por exemplo, quando a satisfação do objetivo resulta de imediato de uma única ação. Outras vezes ela será mais complicada — por exemplo, quando o agente tiver de considerar longas sequências de ações até encontrar um meio de atingir o objetivo. **Busca** (Capítulos 3 a 5) e **planejamento** (Capítulos 10 e 11) são as subáreas da IA dedicados a encontrar sequências de ações que alcançam os objetivos do agente.

Note que a tomada de decisões desse tipo é fundamentalmente distinta das regras condição-ação descritas anteriormente, pelo fato de envolver consideração do futuro, tanto de "O que acontecerá se eu fizer isso e aquilo?" e "Isso me fará feliz?". Nos projetos de agentes reativos, essas informações não são representadas de forma explícita porque as regras internas fazem o mapeamento direto de percepções para ações. O agente reativo freia quando vê luzes de freio. Em princípio, um agente baseado em objetivos poderia raciocinar que, se o carro da frente tem suas luzes de freio acesas, ele diminuirá a velocidade. Dada a forma como o mundo costuma evoluir, a única ação que alcançará o objetivo de não atingir outros carros é frear.

Embora o agente baseado em objetivos pareça menos eficiente, ele é mais flexível porque o conhecimento que apoia suas decisões é representado de maneira explícita e pode ser modificado. Se começar a chover, o agente poderá atualizar seu conhecimento de como seus freios vão operar de modo eficiente; isso fará todos os comportamentos relevantes serem alterados automaticamente para atender às novas condições. Por outro lado, para o agente reativo, teríamos de reescrever muitas regras condição-ação. O comportamento do agente baseado em objetivos pode ser alterado com facilidade para ir a um destino diferente, simplesmente especificando o destino como objetivo. As regras do agente reativo sobre quando fazer curvas e quando seguir em frente só funcionarão para um único destino; todas elas terão de ser substituídas se for preciso ir para algum outro lugar.

2.4.5 Agentes baseados na utilidade

Sozinhos, os objetivos não são realmente suficientes para gerar um comportamento de alta

qualidade na maioria dos ambientes. Por exemplo, existem muitas sequências de ações que levarão o táxi até seu destino (alcançando assim o objetivo), mas algumas são mais rápidas, mais seguras, mais confiáveis ou mais econômicas que outras. Os objetivos simplesmente permitem uma distinção binária crua entre "estados felizes" e "infelizes", enquanto uma medida de desempenho mais geral deve permitir uma comparação entre diferentes estados do mundo, de acordo com o grau exato de felicidade que proporcionariam ao agente. Tendo em vista que "feliz" não soa muito científico, em vez disso, economistas e cientistas da computação usam o termo **utilidade**.[6]

Nós já vimos que uma medida de desempenho atribui uma pontuação para qualquer sequência de estados do ambiente, e assim ela pode distinguir facilmente entre formas mais e menos desejáveis de chegar ao destino do táxi. A função **utilidade do agente** é essencialmente uma internalização da medida de desempenho. Se a função utilidade interna e a medida externa de desempenho estiverem em acordo, um agente que escolhe ações que maximizem a sua utilidade será racional de acordo com a medida de desempenho externa.

Vamos enfatizar novamente que essa não é a *única* maneira de ser racional — já vimos um programa de agente racional para o mundo do aspirador de pó (Figura 2.8) que não tem ideia de qual seja sua função utilidade, mas, como os agentes baseados em objetivos, um agente baseado em utilidade tem muitas vantagens em termos de flexibilidade e de aprendizagem. Além disso, em dois tipos de casos, os objetivos são inadequados, mas um agente baseado em utilidade ainda pode tomar decisões racionais. Primeiro, quando houver objetivos conflitantes, apenas alguns dos quais podem ser alcançados (por exemplo, velocidade e segurança), a função utilidade especifica a escolha apropriada. Segundo, quando há vários objetivos que o agente pode visar e nenhum dos quais pode ser alcançado com certeza, a utilidade proporciona uma maneira em que a probabilidade de sucesso pode ser pesada em relação à importância dos objetivos. Observabilidade parcial e estocasticidade são onipresentes no mundo real e assim, portanto, a tomada de decisão sob incerteza. Tecnicamente falando, um agente racional baseado em utilidade escolhe a ação que maximiza a **utilidade esperada** dos resultados da ação, isto é, a utilidade que o agente espera obter, em média, dadas as probabilidades e as utilidades de cada resultado (o Apêndice A define expectativa mais precisamente). No Capítulo 16, mostraremos que qualquer agente racional deve se comportar *como se* possuísse uma função utilidade cujo valor esperado ele tenta maximizar. Um agente que possui uma função utilidade *explícita* pode tomar decisões racionais por meio de um algoritmo de uso geral que não depende da função utilidade específica que está sendo maximizada. Desse modo, a definição "global" de racionalidade — designando-se como racionais as funções de agentes que têm o melhor desempenho — é transformada em uma restrição "local" sobre projetos de agentes racionais que podem ser expressos em um programa simples.

A estrutura de agente baseado na utilidade aparece na Figura 2.14. Os programas de agentes baseados na utilidade são examinados na Parte IV, em que projetamos agentes de tomada de decisões que devem lidar com a incerteza inerente aos ambientes estocásticos ou parcialmente observáveis.

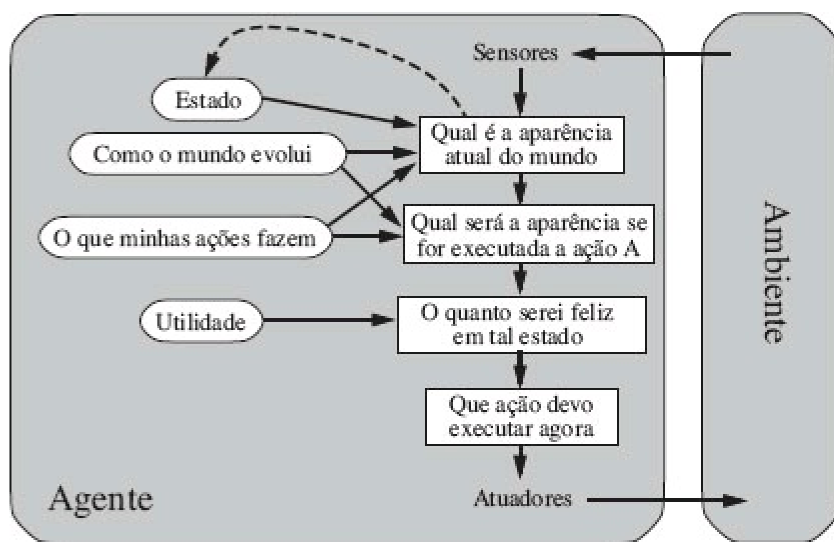


Figura 2.14 Um agente baseado em modelo e orientado para a utilidade. Ele usa um modelo do mundo juntamente com uma função utilidade que mede suas preferências entre estados do mundo. Em seguida, ele escolhe a ação que leva à melhor utilidade esperada, na qual a utilidade esperada é calculada pela média entre todos os estados resultantes possíveis, ponderados pela probabilidade do resultado.

Neste ponto, o leitor pode estar se perguntando: "É assim tão simples? Construímos agentes que maximizam a utilidade esperada e pronto?" É verdade que tais agentes são inteligentes, mas não é simples. Um agente baseado em utilidade tem de modelar e monitorar seu ambiente, tarefas que envolvem grande quantidade de pesquisas sobre percepção, representação, raciocínio e aprendizagem. Os resultados dessa pesquisa preencheram muitos capítulos deste livro. A escolha da maximização de utilidade do curso da ação também é uma tarefa difícil, exigindo algoritmos engenhosos que preencheram outros vários capítulos. Mesmo com esses algoritmos, geralmente o raciocínio perfeito é inatingível na prática por causa da complexidade computacional, mencionada no Capítulo 1.

2.4.6 Agentes com aprendizagem

Descrevemos programas de agentes com vários métodos para selecionar ações. Porém, até agora não explicamos como os programas de agentes *passam a existir*. Em seu famoso ensaio inicial, Turing (1950) considera a ideia de realmente programar suas máquinas inteligentes à mão. Ele estima quanto trabalho isso poderia exigir e conclui que "algum método mais eficiente parece desejável". O método que ele propõe é construir máquinas com aprendizagem e depois ensiná-las. Em muitas áreas de IA, esse é agora o método preferencial para se criar sistemas do estado da arte. O aprendizado tem outra vantagem, como observamos antes: ele permite ao agente operar em ambientes inicialmente desconhecidos e se tornar mais competente do que seu conhecimento inicial sozinho poderia permitir. Nesta seção, introduzimos rapidamente as principais ideias de agentes com aprendizagem. Do começo ao fim do livro, faremos comentários sobre oportunidades e métodos de aprendizado em tipos específicos de agentes. A Parte V estuda com muito maior profundidade os diversos algoritmos de aprendizado propriamente ditos.

Um agente de aprendizado pode ser dividido em quatro componentes conceituais, como mostra a Figura 2.15. A distinção mais importante se dá entre o **elemento de aprendizado**, responsável pela execução de aperfeiçoamentos, e o **elemento de desempenho**, responsável pela seleção de ações externas. O elemento de desempenho é o que antes consideramos como sendo o agente completo: ele recebe percepções e decide sobre ações. O elemento de aprendizado utiliza realimentação do **crítico** sobre como o agente está funcionando e determina de que maneira o elemento de desempenho deve ser modificado para funcionar melhor no futuro.

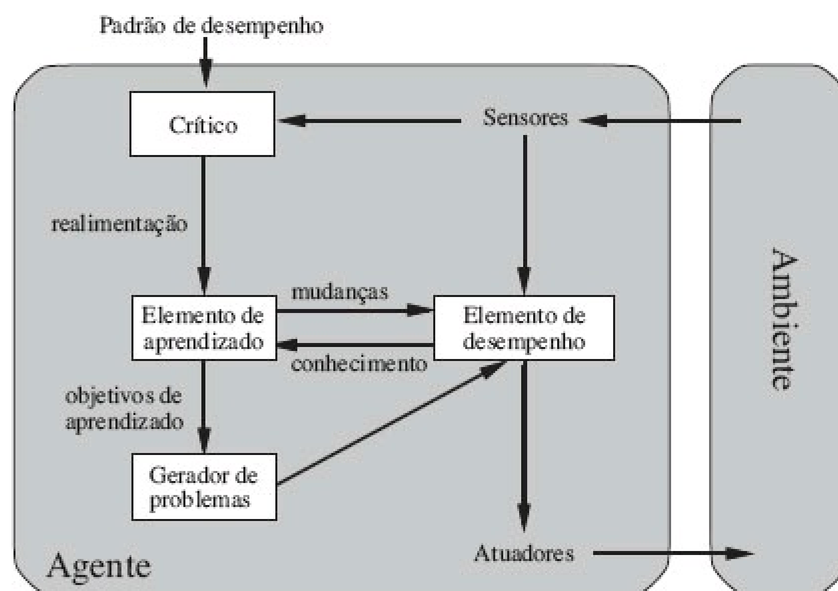


Figura 2.15 Um modelo geral de agentes com aprendizagem.

O projeto do elemento de aprendizado depende muito do projeto do elemento de desempenho. Quando se tenta projetar um agente que aprende certa capacidade, a primeira pergunta não é "Como farei com que ele aprenda isso?", mas "Que tipo de elemento de desempenho meu agente precisará ter para fazer isso depois de ter aprendido como fazê-lo?". Dado um projeto de agente, podem ser construídos mecanismos de aprendizado para otimizar cada parte do agente.

O crítico informa ao elemento de aprendizado como o agente está se comportando em relação a um padrão fixo de desempenho. O crítico é necessário porque as próprias percepções não oferecem nenhuma indicação do sucesso do agente. Por exemplo, um programa de xadrez poderia receber uma percepção indicando que aplicou um xeque-mate em seu oponente, mas o programa precisa de um padrão de desempenho para saber que isso é algo bom; a percepção em si não diz nada sobre isso. É importante que o padrão de desempenho seja fixo. Conceitualmente, deveríamos pensar nele como algo que está totalmente fora do agente porque o agente não deve modificá-lo para ajustá-lo a seu próprio comportamento.

O último componente do agente com aprendizagem é o **gerador de problemas**. Ele é responsável por sugerir ações que levarão a experiências novas e informativas. A questão é que, se o elemento de desempenho tivesse a possibilidade, ele continuaria a realizar as melhores ações, dadas as informações que possui. Porém, se o agente estivesse disposto a realizar uma pequena exploração e executar algumas ações que talvez não fossem ótimas a curto prazo, ele poderia descobrir ações muito melhores a longo prazo. A tarefa do gerador de problemas é sugerir essas ações exploratórias. É isso que os cientistas fazem quando realizam experiências. Galileu não pensava que soltar pedras do alto de uma torre em Pisa teria algum valor em si. Ele não estava tentando quebrar as pedras nem

modificar o cérebro dos pedestres desafortunados. Seu objetivo era modificar seu próprio cérebro, identificando uma teoria melhor sobre o movimento dos objetos.

Para tornar o projeto global mais concreto, vamos voltar ao exemplo do táxi automatizado. O elemento de desempenho consiste em qualquer coleção de conhecimento e procedimentos que o táxi tem para selecionar suas ações de dirigir. O táxi vai para a estrada e dirige, usando esse elemento de desempenho. O crítico observa o mundo e repassa informações ao elemento de aprendizado. Por exemplo, depois que o táxi faz uma rápida mudança para a esquerda cruzando três faixas de tráfego, o crítico observa a linguagem chocante utilizada por outros motoristas. A partir dessa experiência, o elemento de aprendizado é capaz de formular uma regra afirmando que essa foi uma ação ruim, e o elemento de desempenho é modificado pela instalação da nova regra. O gerador de problemas pode identificar certas áreas de comportamento que necessitam de melhorias e sugerir experimentos, como testar os freios em diferentes superfícies de rodagem sob condições distintas.

O elemento de aprendizado pode fazer mudanças em qualquer dos componentes de "conhecimento" mostrados nos diagramas de agentes (Figuras 2.9, 2.11, 2.13 e 2.14). Os casos mais simples envolvem o aprendizado direto a partir da sequência de percepções. A observação de pares de estados sucessivos do ambiente pode permitir ao agente aprender "Como o mundo evolui", e a observação dos resultados de suas ações pode permitir que ele aprenda "O que minhas ações fazem". Por exemplo, se o táxi exercer certa pressão nos freios ao dirigir em uma estrada molhada, ele logo descobrirá qual é a desaceleração realmente alcançada. É claro que essas duas tarefas de aprendizado serão mais difíceis se o ambiente for apenas parcialmente observável.

As formas de aprendizado no parágrafo anterior não precisam ter acesso ao padrão de desempenho externo — de certo modo, o padrão universal é fazer previsões que concordem com a experiência. A situação é um pouco mais complexa no caso de um agente baseado em utilidade que deseja aprender informações de utilidade. Por exemplo, suponha que o agente de direção de táxi não receba dos passageiros nenhuma gorjeta porque o táxi sacolejou muito durante a viagem. O padrão de desempenho externo deve informar ao agente que a falta de gorjetas é uma contribuição negativa para seu desempenho global; desse modo, o agente talvez fosse capaz de aprender que manobras violentas não contribuem para sua própria utilidade. De certo modo, o padrão de desempenho distingue parte da percepção de entrada como uma **recompensa** (ou **penalidade**) que fornece realimentação direta sobre a qualidade do comportamento do agente. Os padrões de desempenho internos como dor e fome em animais podem ser entendidos desse modo. Essa questão é discutida com maior profundidade no Capítulo 21.

Em resumo, os agentes têm uma variedade de componentes, e esses componentes podem ser representados de muitas formas dentro do programa do agente; dessa forma, parece haver grande variedade de métodos de aprendizado. No entanto, existe um único tema unificador. O aprendizado em agentes inteligentes pode ser resumido como um processo de modificação de cada componente do agente, a fim de levar os componentes a um acordo mais íntimo com as informações de realimentação disponíveis, melhorando assim o desempenho global do agente.

2.4.7 Como funcionam os componentes do programa de agente

Descrevemos os programas de agente (em termos de muito alto nível) consistindo de vários componentes cuja função é responder a perguntas como: "Como o mundo está agora?", "Que ações devo tomar?", "O que minhas ações realizam?". A próxima pergunta para um estudante de IA é: "Como funcionam esses componentes?" Levará cerca de mil páginas para começar a responder a essas perguntas apropriadamente, mas queremos chamar a atenção do leitor para algumas distinções básicas entre as várias maneiras como os componentes podem representar o ambiente que o agente habita.

Grosso modo, podemos colocar as representações ao longo de um eixo de complexidade crescente e poder de expressividade — **atômico**, **fatorado** e **estruturado**. Para ilustrar essas ideias, consideremos um componente do agente em particular, como aquele que lida com "O que minhas ações realizam". Esse componente descreve as alterações que podem ocorrer no ambiente como resultado de executar uma ação, e a Figura 2.16 fornece descrições esquemáticas de como as transições devem ser representadas.

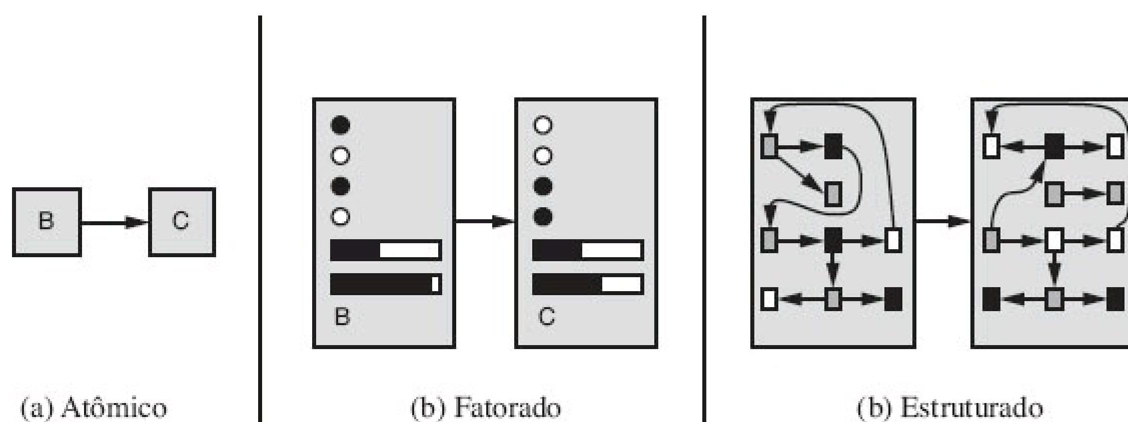


Figura 2.16 Três maneiras de representar os estados e as transições entre eles. (a) Representação atômica: um estado (como B ou C) é uma caixa preta, sem estrutura interna. (b) Representação fatorada: um estado consiste em um vetor de valores de atributos; os valores podem ser booleanos, valores reais ou um conjunto fixo de símbolos. (c) Representação estruturada: um estado inclui objetos; cada um deles pode ter atributos próprios, bem como relacionamentos com outros objetos.

Em uma **representação atômica**, cada estado do mundo é indivisível — não tem estrutura interna. Considere o problema de encontrar um caminho de uma extremidade à outra do país através de alguma sequência de cidades (tratamos esse problema na Figura 3.2). Para resolver esse problema, pode ser suficiente reduzir o estado do mundo apenas para o nome da cidade em que estamos — um único átomo de conhecimento; uma "caixa preta", cuja única propriedade discernível é a de ser idêntico ou diferente de outra caixa preta. Os algoritmos que formam a base de **busca** e de **jogos** (Capítulos 3-5), **Modelos Ocultos de Markov** (Capítulo 15) e os **processos de decisão de Markov** (Capítulo 17) trabalham com representações atômicas ou, pelo menos, tratam as representações *como se* fossem atômicas.

Agora considere uma descrição de maior fidelidade para o mesmo problema, em que precisamos nos preocupar com mais do que apenas a localização atômica em uma cidade ou outra, talvez sendo necessário prestar atenção em quanta gasolina há no tanque, nas coordenadas atuais do GPS, se a luz de advertência do óleo está funcionando, quanto temos de troco para o pedágio, que estação está tocando na rádio, e assim por diante. Uma **representação fatorada** divide cada estado em um

conjunto fixo de **variáveis** ou **atributos**, cada um dos quais pode ter um **valor**. Enquanto dois estados atômicos diferentes não têm nada em comum — são apenas caixas pretas diferentes —, dois estados fatorados diferentes podem compartilhar alguns atributos (como estar em alguma localização GPS específica) e não compartilhar outros (como ter muita ou nenhuma gasolina), o que torna muito mais fácil planejar como ir de um estado para o outro. Com representações fatoradas, também se pode representar a *incerteza* — por exemplo, a ignorância sobre a quantidade de combustível no tanque pode ser representada deixando o atributo em branco. Muitas áreas importantes da IA são baseadas em representações fatoradas, incluindo algoritmos de **satisfação de restrição** (Capítulo 6), **lógica proposicional** (Capítulo 7), **planejamento** (Capítulos 10 e 11), **redes bayesianas** (Capítulos 13-16) e algoritmos de **aprendizado de máquina** nos Capítulos 18, 20 e 21. Para muitos propósitos, é necessário entender que o mundo tem *coisas* que estão *relacionadas* umas com as outras, não apenas variáveis com valores. Por exemplo, podemos notar que um grande caminhão à nossa frente está se movendo em sentido contrário na entrada de carros de uma fazenda de gado leiteiro, mas uma vaca soltou-se e está bloqueando o caminho do caminhão. É pouco provável que uma representação fatorada esteja pré-definida com o atributo *CaminhãoMovendoParaTrásFrenteEntradaFazendaLeiteiraBloqueadoVacaPerdida* com valor *verdadeiro* ou *falso*. Em vez disso, precisaríamos de uma **representação estruturada**, em que objetos, como vacas e caminhões e seus relacionamentos diversos e variados, possam ser explicitamente descritos (ver a Figura 2.16c). Representações estruturadas são base de **bancos de dados relacionais** e **lógica de primeira ordem** (Capítulos 8, 9 e 12), **modelos de probabilidade de primeira ordem** (Capítulo 14), de **aprendizagem baseada em conhecimento** (Capítulo 19) e grande parte da **compreensão da linguagem natural** (Capítulos 22 e 23). Na verdade, quase tudo o que os seres humanos expressam em linguagem natural diz respeito aos objetos e seus relacionamentos.

Como mencionamos anteriormente, o eixo ao longo do qual estão as representações atômica, fatorada e estruturada é o eixo de **expressividade** crescente. Grosseiramente falando, uma representação mais expressiva pode capturar, pelo menos de forma mais concisa, tudo o que algo menos expressivo pode capturar e ainda um pouco mais. Muitas vezes, a linguagem mais expressiva é *muito* mais concisa; por exemplo, as regras do xadrez podem ser escritas em uma página ou duas de uma linguagem com representação estruturada, como lógica de primeira ordem, mas requer milhares de páginas, quando escritas em uma linguagem com representação fatorada, como a lógica proposicional. Por outro lado, o raciocínio e a aprendizagem se tornam mais complexos à medida que aumenta o poder expressivo da representação. Para obter os benefícios das representações expressivas, evitando as suas limitações, pode ser que para o mundo real os sistemas inteligentes necessitem operar simultaneamente em todos os pontos ao longo do eixo de expressividade.

2.5 RESUMO

Este capítulo foi uma espécie de excursão vertiginosa pela IA, que concebemos como a ciência de projeto de agentes. Aqui estão os pontos importantes a serem lembrados:

- Um **agente** é algo que percebe e age em um ambiente. A **função do agente** especifica a ação executada pelo agente em resposta a qualquer sequência de percepções.

- A **medida de desempenho** avalia o comportamento do agente em um ambiente. Um **agente racional** age para maximizar o valor esperado da medida de desempenho, dada a sequência de percepções recebida até o momento.
- Uma especificação de **ambiente de tarefa** inclui a medida de desempenho, o ambiente externo, os atuadores e os sensores. Ao se projetar um agente, o primeiro passo sempre deve ser especificar o ambiente de tarefa de maneira tão completa quanto possível.
- Os ambientes de tarefas variam ao longo de diversas dimensões significativas. Eles podem ser completa ou parcialmente observáveis, agente único ou multiagente, determinísticos ou estocásticos, episódicos ou sequenciais, estáticos ou dinâmicos, discretos ou contínuos e conhecidos ou desconhecidos.
- O **programa do agente** implementa a função do agente. Existe uma variedade de projetos básicos de programas de agentes, refletindo o tipo de informação explicitada e usada no processo de decisão. Os projetos variam em eficiência, síntese e flexibilidade. O projeto apropriado do programa do agente depende da natureza do ambiente.
- Os **agentes reativos simples** respondem diretamente a percepções, enquanto os **agentes reativos baseados em modelos** mantêm o estado interno para controlar aspectos do mundo que não estão evidentes na percepção atual. Os **agentes baseados em objetivos** agem para alcançar seus objetivos, e os **agentes baseados em utilidade** tentam maximizar sua própria "felicidade" esperada.
- Todos os agentes podem melhorar seu desempenho por meio do **aprendizado**.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O papel central da ação na inteligência — a noção de raciocínio prático — remonta pelo menos à época da *Ética a Nicômaco* de Aristóteles. O raciocínio prático também foi o assunto do importante artigo de McCarthy (1958), "Programs with Common Sense". Os campos da robótica e da teoria de controle, por sua própria natureza, se preocupam principalmente com a elaboração de agentes físicos. O conceito de **controlador** em teoria de controle é idêntico ao de um agente em IA. Talvez seja surpreendente o fato de a IA ter se concentrado, durante a maior parte de sua história, em componentes de agentes isolados — sistemas de resposta a perguntas, provadores de teoremas, sistemas de visão, e assim por diante — em lugar de agentes completos. A discussão de agentes no texto de Genesereth e Nilsson (1987) foi uma exceção importante. A visão do agente como um todo é agora extensamente aceita no campo e é tema central em textos recentes (Poole *et al*., 1998; Nilsson, 1998); Padgham e Winikoff, 2004; Jones, 2007).

O Capítulo 1 identificou as raízes do conceito de racionalidade na filosofia e na economia. Em IA, o conceito era de interesse periférico até meados da década de 1980, quando começou a suscitar muitas discussões sobre os próprios fundamentos técnicos do campo. Um artigo de Jon Doyle (1983) previu que o projeto de agentes racionais viria a ser a missão central da IA, enquanto outros tópicos populares acabariam por constituir novas disciplinas.

A atenção cuidadosa às propriedades do ambiente e as suas consequências para o projeto de agentes racionais é mais presente na teoria de controle tradicional — por exemplo, sistemas de

controle clássicos (Dorf e Bishop, 2004; Kirk, 2004) lidam com ambientes completamente observáveis e determinísticos; o controle ótimo estocástico (Kumar e Varaiya, 1986; Bertsekas e Shreve, 2007) trata de ambientes estocásticos parcialmente observáveis, e o controle híbrido (Henzinger e Sastry, 1998; Cassandras e Lygeros, 2006) lida com ambientes que contêm elementos discretos e elementos contínuos. A distinção entre ambientes completa e parcialmente observáveis também é central na literatura de **programação dinâmica** desenvolvida na área de pesquisa operacional (Puterman, 1994), que discutiremos no Capítulo 17.

Os agentes reativos constituíram o modelo fundamental para psicólogos comportamentais como Skinner (1953), que tentou reduzir a psicologia de organismos a mapeamentos de entrada/saída ou estímulo/resposta. O avanço desde o behaviorismo até o funcionalismo em psicologia, que foi pelo menos em parte orientado pela aplicação da metáfora de computadores a agentes (Putnam, 1960; Lewis, 1966), inseriu no cenário o estado interno do agente. A maioria dos trabalhos relacionados à IA vê a ideia de agentes reativos puros com estado como algo demasiado simples para proporcionar grande avanço, mas o trabalho de Rosenschein (1985) e Brooks (1986) questionou essa suposição (ver o Capítulo 25). Nos últimos anos, muito trabalho tem sido dedicado à busca de algoritmos eficientes para controlar ambientes complexos (Hamscher *et al*., 1992; Simon, 2006). O programa Remote Agent (descrito na página 28) que controlava a espaçonave Deep Space One (descrita na página 28) é um exemplo particularmente impressionante (Muscettola *et al*., 1998; Jonsson *et al*., 2000).

Os agentes baseados em objetivos são previstos em tudo desde a visão de Aristóteles do raciocínio prático até os primeiros artigos de McCarthy sobre a IA baseada em lógica. Shakey the Robot (Fikes e Nilsson, 1971; Nilsson, 1984) foi a primeira materialização robótica de um agente lógico baseado em objetivos. Uma análise lógica completa sobre agentes baseados em objetivos foi apresentada em Genesereth e Nilsson (1987), e uma metodologia de programação baseada em objetivos, denominada programação orientada a agentes, foi desenvolvida por Shoham (1993). A abordagem baseada em agente é hoje extremamente popular na engenharia de software (Ciancarini e Wooldridge, 2001). Também se infiltrou na área de sistemas operacionais, onde a **computação autonômica** refere-se a sistemas computacionais e redes que se monitoram e se controlam com um laço de percepção-ação e métodos de aprendizado de máquina (Kephart e Chess, 2003). Observando que uma coleção de programas de agente projetada para trabalhar juntos em um verdadeiro ambiente multiagente, necessariamente apresenta modularidade — os programas não compartilham o estado interno e se comunicam uns com os outros somente através do ambiente —, é comum na área de **sistemas multiagentes** projetar o programa de agente, de um único agente, como uma coleção de subagentes autônomos. Em alguns casos, pode até mesmo ser provado que o sistema resultante devolve as mesmas soluções ótimas que um projeto monolítico.

A visão de agentes baseada em objetivos também domina a tradição da psicologia cognitiva na área de resolução de problemas, começando com o influente trabalho *Human Problem Solving* (Newell e Simon, 1972) e passando por todo o trabalho mais recente de Newell (Newell, 1990). Os objetivos, analisados com maior profundidade como *desejos* (gerais) e *intenções* (atuais), são centrais para a teoria de agentes desenvolvida por Bratman (1987). Essa teoria tem sido influente tanto para a compreensão da linguagem natural quanto para sistemas multiagentes.

Horvitz *et al*. (1988) sugerem especificamente o uso da racionalidade concebida como a

maximização da utilidade esperada, como base para a IA. O texto de Pearl (1988) foi o primeiro em IA a abordar em profundidade a teoria de probabilidade e utilidade; sua exposição de métodos práticos para raciocínio e tomada de decisões sob incerteza talvez tenha sido o maior fator para a rápida mudança em direção a agentes baseados em utilidade nos anos 1990 (ver a Parte IV).

O projeto geral de agentes com aprendizagem representado na Figura 2.15 é clássico na literatura de aprendizagem de máquinas (Buchanan *et al*., 1978; Mitchell, 1997). Exemplos do projeto, materializados em programas, remontam no mínimo ao programa de aprendizado de Arthur Samuel (1959, 1967) para jogar damas. Os agentes com aprendizagem são descritos em profundidade na Parte V.

O interesse em agentes e em projeto de agentes cresceu com rapidez nos últimos anos, em parte devido ao crescimento da Internet e à necessidade percebida de se desenvolver **softbots** automatizados e móveis (Etzioni e Weld, 1994). Documentos relevantes estão reunidos em *Readings in Agents* (Huhns e Singh, 1998), e em *Foundations of Rational Agency* (Wooldridge e Rao, 1999). Textos sobre sistemas multiagentes normalmente fornecem uma boa introdução a muitos aspectos do projeto de agente (Weiss, 2000a; Wooldridge, 2002). Uma série de conferências dedicadas aos agentes começou nos anos 1990, incluindo o Internacional Workshop on Agent Theories, Archictures and Languages (ATAL), a International Conference on Autonomous Agents (AGENTS) e a International Conference on Multi-Agents Systems (ICMAS). Em 2002, essas três se fundiram para formar a International Conference on Autonomous Agents and Multiagent Systems (AAMAS). O periódico *Autonomous Agents and Multi-Agent Systems* foi fundado em 1998. Finalmente, *Dung Beetle Ecology* (Hanski e Cambefort, 1991) fornece grande quantidade de informações interessantes sobre o comportamento de besouros de esterco. O YouTube traz gravações de vídeo inspiradas em suas atividades.

EXERCÍCIOS

2.1 Suponha que a medida de desempenho preocupa-se apenas com os T primeiros passos de tempo do ambiente e ignora tudo a partir de então. Mostre que a ação de um agente racional depende não apenas do estado do ambiente, mas também do passo de tempo que ele alcançou.

2.2 Vamos examinar a racionalidade de várias funções do agente aspirador de pó.

a. Mostre que a função do agente aspirador de pó simples descrito na Figura 2.3 é realmente racional, conforme as suposições listadas na página 38.

b. Descreva uma função de agente racional para o caso em que cada movimento custa um ponto. O programa de agente correspondente exige estado interno?

c. Descreva possíveis projetos de agentes para os casos em que quadrados limpos podem ficar sujos e a geografia do ambiente é desconhecida. Faz sentido para o agente aprender a partir de sua experiência nessas situações? Em caso afirmativo, o que ele deve aprender? Se não, por quê?

2.3 Para cada uma das seguintes afirmações, diga se é verdadeiro ou falso e justifique com exemplos a sua resposta ou com contraexemplos se for o caso.

a. Um agente que detecta apenas informações parciais sobre o estado não pode ser perfeitamente

racional.

b. Existem ambientes de tarefa nos quais nenhum agente reativo puro pode comportar-se racionalmente.

c. Existe um ambiente de tarefa em que todo agente é racional.

d. A entrada para o programa de agente é a mesma que a entrada para a função de agente.

e. Toda função de agente é implementável por uma combinação de programa/máquina.

f. Suponha que um agente selecione sua ação uniformemente ao acaso do conjunto de ações possíveis. Existe um ambiente de tarefa determinista em que esse agente é racional.

g. É possível para um dado agente ser perfeitamente racional em dois ambientes de tarefa distintos.

h. Todo agente é racional em um ambiente não observável.

i. Um agente jogador de pôquer perfeitamente racional nunca perde.

2.4 Para cada uma das seguintes atividades, forneça uma descrição PEAS do ambiente da tarefa e caracterize-o em termos das propriedades listadas na Seção 2.3.2.

- Jogar futebol.
- Explorar os oceanos subterrâneos de Titã.
- Comprar livros usados de IA na Internet.
- Jogar uma partida de tênis.
- Praticar tênis contra uma parede.
- Realizar um salto de altura.
- Licitações de um item em um leilão.

2.5 Defina com suas próprias palavras os termos a seguir: agente, função de agente, programa de agente, racionalidade, autonomia, agente reativo, agente baseado em modelo, agente baseado em objetivos, agente baseado em utilidade, agente com aprendizagem.

2.6 Este exercício explora as diferenças entre funções de agentes e programas de agentes.

a. Pode haver mais de um programa de agente que implemente uma dada função de agente? Dê um exemplo ou mostre por que não é possível.

b. Existem funções de agente que não podem ser implementadas por qualquer programa de agente?

c. Dada uma arquitetura de máquina fixa, cada programa de agente implementa exatamente uma função de agente?

d. Dada uma arquitetura com n bits de armazenamento, quantos programas de agentes distintos são possíveis nessa arquitetura?

e. Suponha manter fixo o programa de agente, mas aumentamos a velocidade da máquina por um fator de dois. Isso muda a função de agente?

2.7 Escreva programas de agente de pseudocódigo para os agentes baseados em objetivos e em utilidade.

Todos os exercícios a seguir estão relacionados à implementação de ambientes e agentes para o

mundo de aspirador de pó.

2.8 Implemente um simulador de ambiente de medição de desempenho para o mundo de aspirador de pó representado na Figura 2.2 e especificado na página 38. Sua implementação deve ser modular, de forma que os sensores, os atuadores e as características do ambiente (tamanho, forma, localização da sujeira etc.) possam ser alterados com facilidade. (*Nota:* Para algumas opções de linguagens de programação e sistemas operacionais, já existem implementações no repositório de código on-line.)

2.9 Implemente um único agente reflexo para o ambiente de vácuo do Exercício 2.8. Execute o ambiente com esse agente para todas as configurações iniciais sujas e localizações do agente possíveis. Registre a nota de desempenho de cada configuração e a nota média global.

2.10 Considere uma versão modificada do ambiente de aspirador de pó do Exercício 2.8, na qual o agente é penalizado com um ponto para cada movimento.

a. Um agente reativo simples pode ser perfeitamente racional para esse ambiente? Explique.

b. E um agente reativo com estado? Projete tal agente.

c. Como suas respostas para os itens **a** e **b** mudarão se as percepções do agente fornecerem o *status* limpo/sujo de cada quadrado no ambiente?

2.11 Considere uma versão modificada do ambiente de aspirador de pó do Exercício 2.8, na qual a geografia do ambiente — extensão, limites e obstáculos — é desconhecida, como também a configuração inicial de sujeira (o agente também pode se mover *Acima* e *Abaixo*, além de *Esquerda* e *Direita*).

a. Um agente reativo simples pode ser perfeitamente racional para esse ambiente? Explique.

b. Um agente reativo simples com uma função de agente *aleatório* pode superar um agente reativo simples? Projete tal agente e faça a medição de seu desempenho em vários ambientes.

c. Você poderia projetar um ambiente no qual seu agente aleatório tenha um desempenho muito ruim? Mostre seus resultados.

d. Um agente reativo com estado pode superar um agente reativo simples? Projete tal agente e faça a medição de seu desempenho em vários ambientes. Você pode projetar um agente racional desse tipo?

2.12 Repita o Exercício 2.11 para o caso em que o sensor de posição é substituído por um sensor de "impacto" que detecta as tentativas realizadas pelo agente para se mover para um obstáculo ou cruzar os limites do ambiente. Suponha que o sensor de impacto pare de funcionar; como o agente deverá se comportar?

2.13 Os ambientes de aspiradores de pó de todos os exercícios anteriores eram determinísticos. Descreva possíveis programas de agentes para cada uma das versões estocásticas listadas a seguir:

a. Lei de Murphy: durante 25% do tempo, a ação de *Aspirar* falha ao limpar o chão se ele está sujo e deposita sujeira no chão se ele está limpo. De que maneira seu programa de agente é afetado se o sensor de sujeira fornece a resposta errada durante 10% do tempo?

b. Crianças pequenas: em cada período de tempo, cada quadrado limpo tem uma chance de 10% de se tornar sujo. Você poderia apresentar um projeto de agente racional para esse caso?

[1] Se o agente utilizasse alguma aleatoriedade para escolher suas ações, teríamos de experimentar cada sequência muitas vezes para identificar a probabilidade de cada ação. Talvez alguém considere a atuação aleatória bastante tola, mas veremos mais adiante, neste capítulo, que ela pode ser muito inteligente.

[2] Veja N. Henderson, “New door latches urged for Boeing 747 jumbo jets”, *Washington Post*, 24 de agosto de 1989.

[3] A palavra “sequencial” também é usada em ciência da computação como antônimo de “paralelo”. Os dois significados não têm qualquer correlação.

[4] Existem outras opções para a estrutura do programa do agente; por exemplo, poderíamos fazer os programas dos agentes serem **coprocedimentos** que são executados de forma assíncrona com o ambiente. Cada um desses procedimentos tem uma porta de entrada e uma porta de saída, e consiste em um laço que lê a porta de entrada em busca de percepções e grava ações na porta de saída.

[5] Também chamadas **regras situação-ação**, **regras de produção** ou **regras se-então**.

[6] A palavra “utilidade” aqui se refere à “qualidade de ser útil,” não à empresa elétrica ou de rede de água.

PARTE II

Resolução de problemas

CAPÍTULO 3

Resolução de problemas por meio de busca

Em que vemos como um agente pode encontrar uma sequência de ações que alcança seus objetivos quando nenhuma ação isolada é capaz de fazê-lo.

Os agentes mais simples examinados no Capítulo 2 eram os agentes reativos, que baseiam suas ações em um mapeamento direto de estados em ações. Tais agentes não podem operar bem em ambientes para os quais esse mapeamento seria grande demais para se armazenar e levaria muito tempo para se aprender. Por outro lado, os agentes baseados em objetivos, consideram ações futuras e o quanto seus resultados são desejáveis.

Este capítulo descreve um tipo de agente baseado em objetivo chamado **agente de resolução de problemas**. Os agentes de resolução de problemas utilizam representações **atômicas**, conforme descrito na Seção 2.4.7, ou seja, os estados do mundo são considerados como um todo, sem estrutura interna visível para os algoritmos de resolução de problemas. Os agentes baseados em objetivos que utilizam representações **fatoradas** ou **estruturadas** mais avançadas, geralmente são chamados de agentes de planejamento e serão discutidos nos Capítulos 7 e 10.

Nossa discussão sobre a resolução de problemas começa com uma definição precisa dos **problemas** e de suas **soluções** e fornece vários exemplos para ilustrar essas definições. Em seguida, descrevemos vários algoritmos de busca de propósito geral que podem ser utilizados para resolver esses problemas. Veremos diversos algoritmos de busca **sem informação** — algoritmos para os quais não se fornece nenhuma informação sobre o problema a não ser sua definição. Embora alguns desses algoritmos possam resolver qualquer problema solucionável, nenhum deles pode fazê-lo de forma eficiente. Os algoritmos de busca **informada**, por outro lado, podem ter sucesso a partir de alguma orientação sobre onde procurar soluções.

Neste capítulo, nos limitaremos ao tipo mais simples de ambiente de tarefa, para o qual a solução para um problema é sempre uma *sequência fixa* de ações. O caso mais geral, no qual as ações futuras do agente podem variar dependendo de percepções futuras, será tratado no Capítulo 4.

Este capítulo utiliza os conceitos de complexidade assintótica (isto é, a notação O ()) e NP-completeza. Os leitores que não estão familiarizados com esses conceitos deverão consultar o Apêndice A.

3.1 AGENTES DE RESOLUÇÃO DE PROBLEMAS

Os agentes inteligentes devem maximizar sua medida de desempenho. Como mencionamos no Capítulo 2, esse objetivo é às vezes simplificado se o agente pode adotar um **objetivo** que deseja satisfazer. Primeiro, vamos examinar por que e como um agente poderia realizar isso.

Imagine um agente na cidade de Arad, na Romênia, aproveitando uma viagem de férias. A medida de desempenho do agente contém muitos fatores: ele quer melhorar seu bronzeado, melhorar seu conhecimento do idioma romeno, ver as paisagens, apreciar a vida noturna, evitar ressacas, e assim por diante. O problema de decisão é complexo e envolve muitos compromissos e leitura cuidadosa de guias de viagem. Agora, suponha que o agente tenha uma passagem não reembolsável para partir de Bucareste na manhã seguinte. Nesse caso, faz sentido para o agente adotar o **objetivo** de chegar a Bucareste. Os cursos de ação que não chegam a Bucareste a tempo podem ser rejeitados sem consideração adicional, e o problema de decisão do agente fica bastante simplificado. Os objetivos ajudam a organizar o comportamento, limitando o que o agente está tentando alcançar e, consequentemente, as ações que ele precisa considerar. **A formulação de objetivos**, baseada na situação atual e na medida de desempenho do agente, é o primeiro passo para a resolução de problemas.

Vamos considerar que um objetivo seja um conjunto de estados do mundo — exatamente os estados em que o objetivo é satisfeito. A tarefa do agente é descobrir como agir, agora e no futuro, para que atinja o estado objetivo. Antes de poder fazer isso, ele precisa decidir (ou precisamos decidir por ele) que tipo de ações e estados deveria considerar. Se tentasse considerar ações ao nível de "mover o pé esquerdo para a frente uma polegada" ou "girar o volante um grau para a esquerda", o agente provavelmente nunca conseguiria sair do estacionamento, quanto mais chegar a Bucareste, porque nesse nível de detalhe existe muita incerteza no mundo e haveria muitos passos para se chegar a uma solução. A **formulação de problemas** é o processo de decidir que ações e estados devem ser considerados, dado um objetivo. Examinaremos mais adiante os detalhes desse processo. No momento, vamos supor que o agente vai considerar ações no nível de dirigir desde uma cidade importante até outra. Cada estado corresponderá, portanto, a estar em uma determinada cidade.

Nosso agente agora adotou o objetivo de dirigir para Bucareste e está considerando para onde ir a partir de Arad. Existem três estradas que saem de Arad, uma em direção a Sibiu, uma para Timisoara e uma para Zerind. Nenhuma delas atinge o objetivo; assim, a menos que o agente esteja muito familiarizado com a geografia da Romênia, ele não saberá que estrada deve seguir.[1] Em outras palavras, o agente não saberá qual das ações possíveis é a melhor porque não conhece ainda o suficiente sobre o estado que resulta da execução de cada ação. Se o agente não tiver nenhuma informação adicional, ou seja, se o ambiente for **desconhecido** no sentido definido na Seção 2.3, ele não terá escolha a não ser tentar uma das ações de forma aleatória. Essa triste situação é discutida no Capítulo 4.

☞ Porém, suponha que o agente tenha um mapa da Romênia. A finalidade de um mapa é fornecer ao agente informações sobre os estados em que ele próprio pode visitar e sobre as ações que ele pode executar. O agente pode usar essas informações para considerar estágios *subsequentes* de uma

jornada hipotética passando por cada uma das três cidades, procurando descobrir um percurso que eventualmente chegue a Bucareste. Depois de encontrar um caminho no mapa de Arad até Bucareste, ele poderá alcançar seu objetivo executando as ações de dirigir que correspondem aos passos da viagem. Em geral, *um agente com várias opções imediatas de valor desconhecido pode decidir o que fazer examinando primeiro ações futuras que levam eventualmente a estados de valor conhecido.*

Para ser mais específico sobre o que entendemos por "examinar as ações futuras", temos que ser mais específicos sobre as propriedades do ambiente, como definido na Seção 2.3. Por ora, assumiremos que o ambiente é **observável**, de modo que o agente sempre conhecerá o estado atual. Para o agente dirigir na Romênia, é razoável supor que cada cidade no mapa tenha uma placa indicando o nome da cidade aos motoristas que chegam. Assumiremos também que o ambiente seja **discreto**; assim, em qualquer estado dado, haverá apenas um número finito de ações para escolher. Isso é verdadeiro para a navegação na Romênia, pois cada cidade é conectada a um pequeno número de outras cidades. Supomos que o ambiente seja **conhecido**; assim, o agente sabe quais estados serão alcançados em cada ação (para problemas de navegação, ter um mapa preciso é suficiente para atender a essa condição). Finalmente, assumiremos que o ambiente seja **determinístico**; portanto, cada ação tem exatamente um resultado. Em condições ideais, isso é verdadeiro para o agente na Romênia — significa que, se escolher dirigir de Arad para Sibiu, acabará em Sibiu. Certamente as condições nem sempre são ideais, como mostraremos no Capítulo 4.

Sob essas premissas, a solução para qualquer problema é uma sequência fixa de ações. "Claro!", pode-se dizer, "o que mais poderia ser?" Bem, em geral, poderia ser uma estratégica ramificada que recomenda ações diferentes no futuro dependendo das percepções recebidas. Por exemplo, em condições menos que ideais, o agente poderá planejar dirigir de Arad para Sibiu e depois para Rimnicu Vilcea, mas pode também necessitar de um plano de contingência, caso acidentalmente chegue a Zerind, em vez de em Sibiu. Felizmente, se o agente souber o estado inicial e o ambiente for conhecido e determinístico, saberá exatamente onde estará após a primeira ação e o que vai perceber. Uma vez que, após a primeira ação, é possível receber apenas uma percepção, a solução poderá especificar apenas uma segunda ação possível, e assim por diante.

Esse processo de procurar por tal sequência de ações que alcançam o objetivo é chamado de **busca**. Um algoritmo de busca recebe um problema como entrada e devolve uma **solução** sob a forma de uma sequência de ações. Depois que uma solução é encontrada, as ações recomendadas podem ser executadas. Isso se chama fase de **execução**. Desse modo, temos um simples projeto de "formular, buscar, executar" para o agente, como mostra a Figura 3.1. Depois de formular um objetivo e um problema a resolver, o agente chama um procedimento de busca para resolvê-lo. Em seguida, ele utiliza a solução para orientar suas ações, fazendo o que a solução recomendar como a próxima ação — em geral, a primeira ação da sequência — e então removendo esse passo da sequência. Depois que a solução for executada, o agente formulará um novo objetivo.

Observe que, enquanto o agente está executando a sequência de solução, *ele ignora sua percepção ao escolher uma ação porque sabe com antecedência qual será*. Um agente que realiza seus planos com os olhos fechados, por assim dizer, deve estar completamente certo do que está acontecendo. Os teóricos de controle chamam isso de um sistema de **malha aberta**, pois ignorar a percepção quebra o laço entre o agente e o ambiente.

Primeiramente, descreveremos o processo de formulação de problemas e depois dedicaremos a parte principal do capítulo a diversos algoritmos para a função BUSCA. Não discutiremos mais neste capítulo as funções ATUALIZAR-ESTADO e FORMULAR-PROBLEMAS.

3.1.1 Problemas e soluções bem definidos

Um **problema** pode ser definido formalmente por cinco componentes:

- O **estado inicial** em que o agente começa. Por exemplo, o estado inicial do nosso agente na Romênia poderia ser descrito como *Em*(*Arad*).

```
função AGENTE-DE RESOLUÇÃO-DE-PROBLEMAS-SIMPLES(percepção) retorna uma ação
   persistente: seq, uma sequência de ações, inicialmente vazia
                estado, alguma descrição do estado atual do mundo
                objetivo, um objetivo, inicialmente nulo
                problema, uma formulação de problema
   estado ← ATUALIZAR-ESTADO(estado, percepção)
   se seq está vazia então faça
      objetivo ← FORMULAR-OBJETIVO(estado)
      problema ← FORMULAR-PROBLEMA(estado, objetivo)
      seq ← BUSCA(problema)
      se seq = falhar então retorne uma ação nula
   ação ← PRIMEIRO(seq)
   seq ← RESTO(seq)
   retornar ação
```

Figura 3.1 Um agente simples de resolução de problemas. Primeiro, ele formula um objetivo e um problema, busca uma sequência de ações que resolvem o problema e depois executa as ações, uma de cada vez. Quando essa sequência se completa, ele formula outro objetivo e recomeça.

- Uma descrição das **ações** possíveis que estão disponíveis para o agente. Dado um estado particular *s*, AÇÕES*(s)* devolve um conjunto de ações que podem ser executadas em *s*. Dizemos que cada uma dessas ações é **aplicável** em *s*. Por exemplo, a partir do estado *Em(Arad),* as ações aplicáveis são: {Ir(*Sibiu*), Ir(*Timisoara*), Ir(*Zerind*)}.
- Uma descrição do que cada ação faz; o nome formal é **modelo de transição**, especificado por uma função RESULTADO(*s, a*) que devolve o estado que resulta de executar uma ação *a* em estado *s*. Usamos também o termo **sucessor** para nos referirmos a qualquer estado acessível a partir de determinado estado por uma única ação.[2] Por exemplo, temos RESULTADO (Em (*Arad*), Ir (*Zerind*)) = Em (*Zerind*). Juntos, o estado inicial, as ações e o modelo de transição definem implicitamente o **espaço de estados** do problema — o conjunto de todos os estados acessíveis a partir do estado inicial, por qualquer sequência de ações. O espaço de estados

forma uma rede dirigida ou um **grafo** em que os nós são estados e os arcos entre os nós são ações (o mapa da Romênia mostrado na Figura 3.2 pode ser interpretado como um grafo do espaço de estados se visualizarmos cada estrada como duas possíveis ações de dirigir, uma para cada sentido). Um **caminho** no espaço de estados é uma sequência de estados conectados por uma sequência de ações.

- O **teste de objetivo**, que determina se um estado é um estado objetivo. Às vezes existe um conjunto explícito de estados objetivo possíveis, e o teste simplesmente verifica se o estado dado é um deles. O objetivo do agente na Romênia é o conjunto unitário {Em(*Bucareste*)}. Algumas vezes, o objetivo é especificado por uma propriedade abstrata, e não por um conjunto de estados explicitamente enumerado. Por exemplo, no xadrez, o objetivo é alcançar um estado chamado “xeque-mate”, em que o rei do oponente está sob ataque e não consegue escapar.

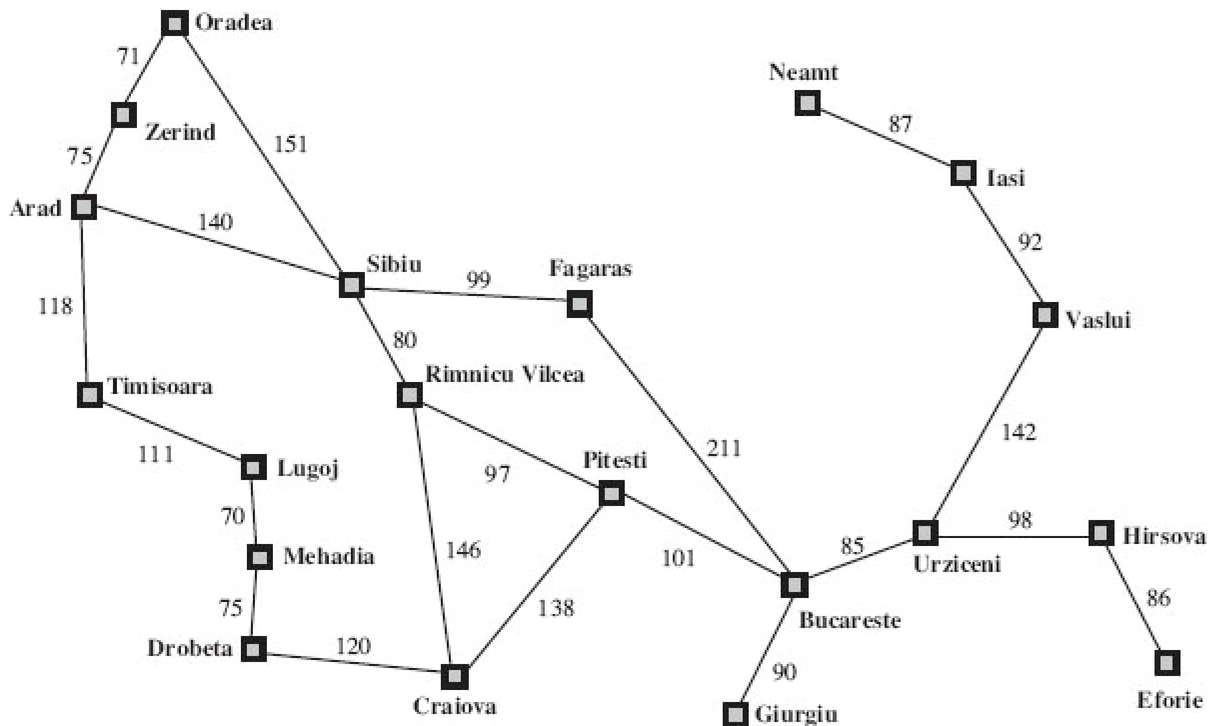


Figura 3.2 Mapa rodoviário simplificado de parte da Romênia.

- Uma função de **custo de caminho** que atribui um custo numérico a cada caminho. O agente de resolução de problemas escolhe uma função de custo que reflete sua própria medida de desempenho. Para o agente que tenta chegar a Bucareste, o tempo é essencial e, assim, o custo de um caminho poderia ser seu comprimento em quilômetros. Neste capítulo, supomos que o custo de um caminho pode ser descrito como a soma dos custos das ações individuais ao longo do caminho.[3] O **custo do passo** de adotar a ação *s* para alcançar o estado *s´* é denotado por *c(s, a, s ´)*. Os custos dos passos para a Romênia são mostrados na Figura 3.2 como distâncias de rotas. Vamos supor que os custos dos passos sejam não negativos.[4]

Os elementos precedentes definem um problema e podem ser reunidos em uma única estrutura de dados que é fornecida como entrada para um algoritmo de resolução de problemas. Uma **solução** para um problema é um caminho desde o estado inicial até um estado objetivo. A qualidade da solução é medida pela função de custo de caminho, e uma **solução ótima** tem o menor custo de caminho entre todas as soluções.

3.1.2 Formulação de problemas

Na seção anterior, propusemos uma formulação do problema de chegar a Bucareste em termos do estado inicial, ações, modelo de transição, teste de objetivo e custo de caminho. Essa formulação parece razoável, mas é ainda um *modelo* — uma descrição matemática abstrata — e não do mundo real. Compare a descrição do estado simples que escolhemos, *Em(Arad),* a uma viagem real cruzando o país, em que o estado do mundo inclui muitos itens: os companheiros de viagem, o programa de rádio atual, a paisagem vista da janela, a proximidade de policiais, a distância até a próxima parada para descanso, as condições da estrada, o tempo, e assim por diante. Todas essas considerações são omitidas de nossas descrições de estados porque são irrelevantes para o problema de encontrar uma rota para Bucareste. O processo de remover detalhes de uma representação é chamado **abstração**.

Além de abstrair a descrição do estado, devemos abstrair as próprias ações. Uma ação de direção tem muitos efeitos. Além de mudar a posição do veículo e de seus ocupantes, ela gasta tempo, consome combustível, gera poluição e muda o agente (como se costuma dizer, viajar é expandir seus horizontes). Nossa formulação leva em conta apenas a mudança de posição. Além disso, existem muitas ações que omitiremos por completo: ligar o rádio, olhar pela janela, diminuir a velocidade ao se aproximar de policiais, e assim por diante. É claro que não especificamos ações no nível de "girar o volante um grau para a esquerda".

Podemos ser mais precisos quanto à definição do nível apropriado de abstração? Pense nos estados abstratos e nas ações que escolhemos como correspondentes a grandes conjuntos de estados detalhados do mundo e sequências detalhadas de ações. Agora, considere uma solução para o problema abstrato: por exemplo, o caminho de Arad para Sibiu para Rimnicu Vilcea para Pitesti para Bucareste. Essa solução abstrata corresponde a um grande número de caminhos mais detalhados. Como exemplo, poderíamos dirigir com o rádio ligado entre Sibiu e Rimnicu Vilcea, e depois desligá-lo pelo restante da viagem. A abstração será *válida* se pudermos expandir qualquer solução abstrata em uma solução no mundo mais detalhado; uma condição suficiente é que, para cada estado detalhado como "em Arad", existe um caminho detalhado para algum estado como "em Sibiu", e assim por diante.[5] A abstração é *útil* se a execução de cada uma das ações na solução é mais fácil que o problema original; nesse caso, elas são fáceis o bastante para poderem ser executadas sem busca ou planejamento adicional para qualquer agente de direção. A escolha de uma boa abstração envolve, portanto, a remoção da maior quantidade possível de detalhes, enquanto se preserva a validade e se assegura que as ações abstratas são fáceis de executar. Se não fosse a habilidade de elaborar abstrações úteis, os agentes inteligentes seriam completamente sufocados pelo mundo real.

3.2 EXEMPLOS DE PROBLEMAS

A abordagem de resolução de problemas é aplicada a uma ampla série de ambientes de tarefas. Listamos aqui alguns dos mais conhecidos, fazendo distinção entre *problemas de mundos simplificados* e *problemas do mundo real.* Um **mundo simplificado** se destina a ilustrar ou exercitar diversos métodos de resolução de problemas. Ele pode ter uma descrição concisa e exata e ser,

portanto, utilizável por diferentes pesquisadores para comparar o desempenho dos algoritmos. Um **problema do mundo real** é aquele cujas soluções de fato interessam às pessoas. Tais problemas tendem a não apresentar uma única descrição consensual, mas tentaremos dar uma ideia geral de suas formulações.

3.2.1 Problemas de mundo simplificado

O primeiro exemplo que examinaremos é o mundo do aspirador de pó introduzido inicialmente no Capítulo 2 (veja a Figura 2.2.). Ele pode ser formulado como um problema da seguinte forma:

- **Estados**: O estado é determinado tanto pela posição do agente como da sujeira. O agente está em uma entre duas posições, cada uma das quais pode conter sujeira ou não. Desse modo, há $2 \times 2^2 = 8$ estados do mundo possíveis. Um ambiente mais amplo com *n* posições tem $n \cdot 2^n$ estados.
- **Estado inicial**: Qualquer estado pode ser designado como o estado inicial.
- **Ações**: Nesse ambiente simples, cada estado tem apenas três ações: *Esquerda, Direita* e *Aspirar.* Ambientes mais amplos podem também incluir *Em Cima* e *Embaixo.*
- **Modelo de transição**: As ações têm seus efeitos esperados, a não ser as ações: mover para a *Esquerda* no quadrado mais à esquerda, mover para *a Direita*, no quadrado mais à direita, e *Aspirar*, no quadrado limpo, que não tem nenhum efeito. O espaço de estados completo é mostrado na Figura 3.3.
- **Teste de objetivo:** Verifica se todos os quadrados estão limpos.
- **Custo de caminho:** Cada passo custa 1 e, assim, o custo do caminho é o número de passos do caminho.

Comparado com o mundo real, esse problema de mundo simplificado tem posições discretas, sujeira discreta, limpeza confiável e nunca se torna sujo depois de limpo. No Capítulo 4 relaxaremos algumas dessas suposições.

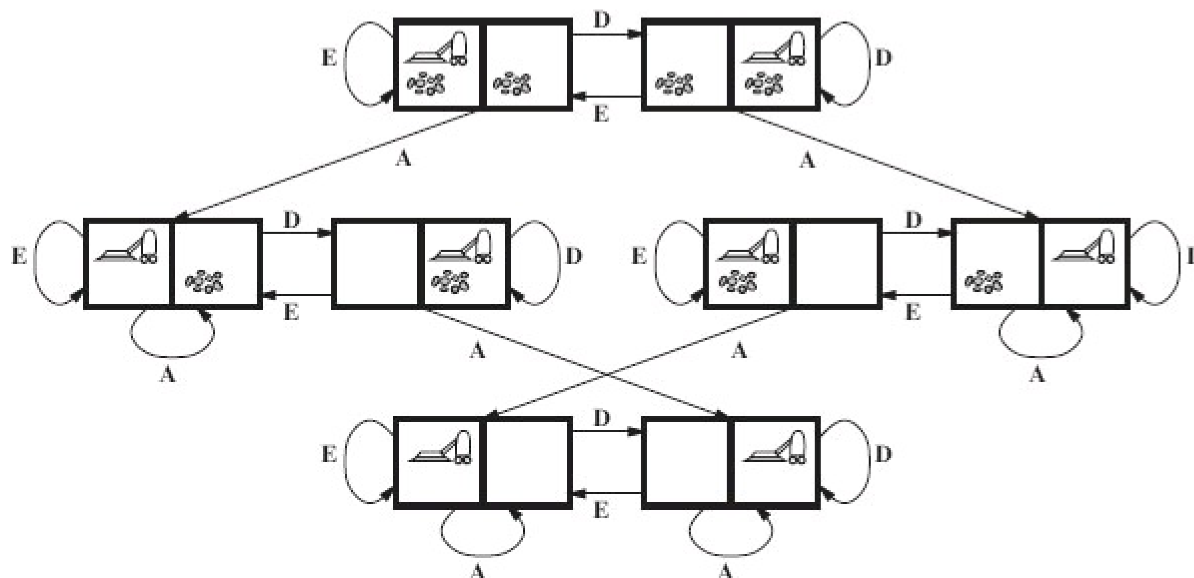


Figura 3.3 O espaço de estados para o mundo do aspirador de pó. Os arcos denotam ações: E = *Esquerda*, D = *Direita*, A = *Aspirar*.

O **quebra-cabeça de oito peças** (veja um exemplo na Figura 3.4), consiste de um tabuleiro 3 × 3 com oito peças numeradas e um quadrado vazio. Uma peça adjacente ao quadrado vazio pode deslizar para esse quadrado. O objetivo é alcançar um estado objetivo especificado, como o do lado direito da figura. A formulação-padrão é dada por:

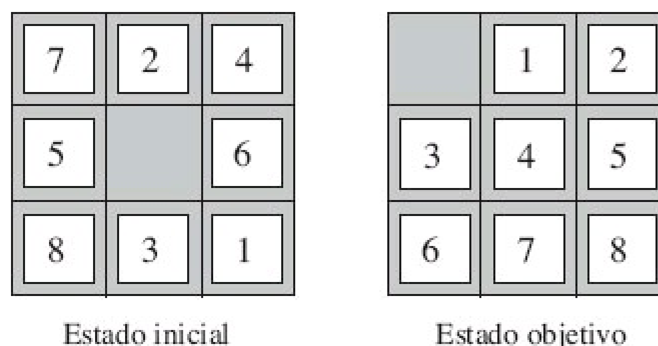


Figura 3.4 Uma instância típica do quebra-cabeça de oito peças.

- **Estados:** Uma descrição de estado especifica a posição de cada uma das oito peças e do quadrado vazio em um dos nove quadrados.
- **Estado inicial**: Qualquer estado pode ser designado como o estado inicial. Observe que qualquer objetivo específico pode ser alcançado a partir de exatamente metade dos estados iniciais possíveis (Exercício 3.4).
- **Ações:** A formulação mais simples define as ações como movimentos do quadrado vazio *Esquerda*, *Direita*, *Para Cima ou Para Baixo*. Pode haver subconjuntos diferentes desses, dependendo de onde estiver o quadrado vazio.
- **Modelo de transição**: Dado um estado e ação, ele devolve o estado resultante; por exemplo, se aplicarmos *Esquerda* para o estado inicial na Figura 3.4, o estado resultante terá comutado o 5 e o branco.
- **Teste de objetivo**: Verifica se o estado corresponde à configuração de estado objetivo mostrada na Figura 3.4 (são possíveis outras configurações de objetivos).
- **Custo de caminho**: Cada passo custa 1 e,assim, o custo do caminho é o número de passos do caminho.

Que abstrações incluímos aqui? As ações são reduzidas a seus estados iniciais e finais, ignorando-se as posições intermediárias por onde uma peça está deslizando. Abstraímos ações como sacudir o tabuleiro quando as peças ficam presas e descartar a extração das peças com uma faca e colocá-las de volta no tabuleiro. Ficamos com uma descrição das regras do quebra-cabeça, evitando todos os detalhes de manipulações físicas.

O quebra-cabeça de oito peças pertence à família de **quebra-cabeças de blocos deslizantes**, usados com frequência como problemas de teste para novos algoritmos de busca em IA. Essa família é conhecida como NP-completa; assim, ninguém espera encontrar métodos significativamente melhores no pior caso que os algoritmos de busca descritos neste capítulo e no próximo. O quebra-cabeça de oito peças tem $9!/2 = 181.440$ estados acessíveis e é resolvido com facilidade. O quebra-cabeça de 15 peças (em um tabuleiro 4 × 4) tem aproximadamente 1,3 trilhão de estados, e instâncias aleatórias podem ser resolvidas de forma ótima em alguns milissegundos pelos melhores algoritmos de busca. O quebra-cabeça de 24 peças (em um tabuleiro 5 × 5) tem cerca de 10^{25} estados, e

instâncias aleatórias demandam muitas horas para resolver de forma ótima.

O objetivo do **problema de oito rainhas** é posicionar oito rainhas em um tabuleiro de xadrez de tal forma que nenhuma rainha ataque qualquer outra (uma rainha ataca qualquer peça situada na mesma linha, coluna ou diagonal). A Figura 3.5 mostra uma tentativa de solução que falhou: a rainha na coluna mais à direita é atacada pela rainha do canto superior esquerdo.

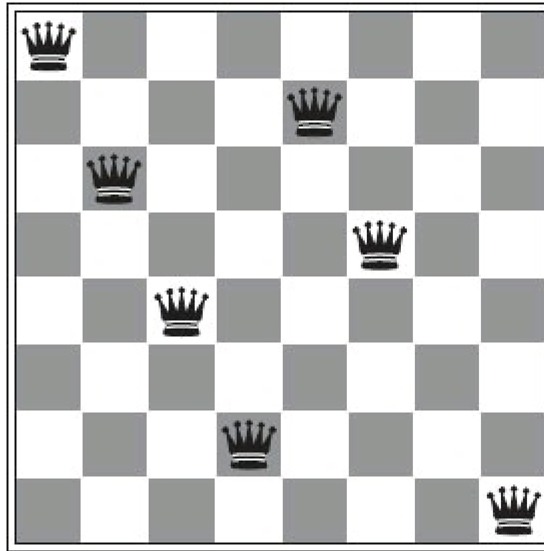


Figura 3.5 Uma quase solução para o problema das oito rainhas (a solução fica como exercício).

Embora existam algoritmos de propósito específico eficientes para esse problema e para toda a família de *n* rainhas, ele continua a ser um problema interessante de teste para algoritmos de busca. Há dois tipos principais de formulações. Uma **formulação incremental** envolve operadores que *ampliam* a descrição dos estados, iniciando com um estado vazio. Para o problema de oito rainhas, isso significa que cada ação acrescenta uma rainha ao estado. Uma **formulação de estados completos** começa com todas as oito rainhas e as desloca pelo tabuleiro. Em qualquer caso, o custo de caminho não tem nenhum interesse porque apenas o estado final é importante. A primeira formulação incremental que se poderia experimentar é:

- **Estados:** Qualquer disposição de 0-8 rainhas no tabuleiro é um estado.
- **Estado inicial**: Nenhuma rainha no tabuleiro.
- **Ações:** Colocar uma rainha em qualquer quadrado vazio.
- **Modelo de transição:** Devolver uma rainha adicionada em qualquer quadrado específico no tabuleiro.
- **Teste de objetivo:** Oito rainhas estão no tabuleiro e nenhuma é atacada.

Nessa formulação, temos $64 \cdot 63 \cdots 57 \approx 1{,}8 \times 10^{14}$ sequências possíveis para investigar. Uma formulação melhor proibiria a colocação de uma rainha em qualquer quadrado que já estiver sob ataque:

- **Estados:** Todas as possíveis disposições de *n* rainhas ($0 \leq n \leq 8$), uma por coluna nas *n* colunas mais à esquerda, sem que nenhuma rainha ataque outra.
- **Ações:** Adicione uma rainha a qualquer quadrado na coluna vazia mais à esquerda, de tal modo

que ela não seja atacada por qualquer outra rainha.

Essa formulação reduz o espaço de estados de oito rainhas de $1{,}8 \times 10^{14}$ para apenas 2.057, e as soluções são fáceis de encontrar. Por outro lado, para 100 rainhas, a formulação inicial tem aproximadamente 10^{400} estados, enquanto a formulação melhorada tem cerca de 10^{52} estados (Exercício 3.5) — uma grande melhoria, mas não o suficiente para tornar o problema tratável. A Seção 4.1 descreve a formulação de estados completos, e o Capítulo 6 apresenta um algoritmo simples que resolve com facilidade até mesmo o problema de um milhão de rainhas.

Nosso problema de mundo simplificado final foi inventado por Donald Knuth (1964) e ilustra como podem surgir espaços de estados infinitos. Knuth conjecturou que, começando com o número 4, uma sequência de fatoriais, raiz quadrada e operações de arredondamento para baixo é possível chegar a qualquer inteiro positivo desejado. Por exemplo, podemos chegar a 5 de 4 da seguinte forma:

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5\,.$$

A definição do problema é muito simples:

- **Estados:** Números positivos.
- **Estado inicial:** 4.
- **Ações:** Aplicar fatorial, raiz quadrada ou operação arredondamento para baixo (somente fatorial de números inteiros).
- **Modelo de transição:** Como dado pelas definições matemáticas das operações.
- **Teste de objetivo:** Estado é o inteiro positivo desejado.

Do que sabemos, não há limite do quão grande é um número que pode ser construído no processo de alcançar determinado objetivo, por exemplo, o número 620.448.401.733.239.439.360.000 é gerado na expressão para 5, de modo que o espaço de estados para esse problema é infinito. Tal espaço de estados surge com frequência em tarefas que envolvem a geração de expressões matemáticas, circuitos, provas, programas e outros objetos definidos de forma recursiva.

3.2.2 Problemas do mundo real

Já vimos como o **problema de roteamento** é definido em termos de posições especificadas e transições ao longo de ligações entre eles. Uma variedade de aplicativos utiliza algoritmos de roteamento. Alguns, como sites da Web e sistemas para automóveis que fornecem instruções de direção, são extensões relativamente simples do exemplo da Romênia. Outros, como sistemas de roteamento de fluxos de vídeo em redes de computadores, planejamento de operações militares e planejamento de viagens aéreas, implicam especificações muito mais complexas. Considere os problemas de viagens aéreas que devem ser resolvidos através de um site da Web de planejamento de viagem:

- **Estados**: Cada estado, obviamente, inclui uma posição (por exemplo, um aeroporto) e o tempo presente. Além disso, como o custo de uma ação (um segmento de voo) pode depender de segmentos anteriores, das suas bases de tarifa e da sua condição de ser nacional ou internacional, o estado deverá ter registro de informações adicionais sobre esses aspectos "históricos".
- **Estado inicial**: Especificado pela pergunta do usuário.
- **Ações:** Pegar qualquer voo a partir da posição atual, em qualquer classe de assento, partindo após o instante atual, deixando tempo suficiente para translado no aeroporto, se necessário.
- **Modelo de transição**: O estado resultante de pegar um voo terá o destino do voo como a posição atual e a hora de chegada do voo como o instante atual.
- **Teste de objetivo**: Estamos no destino final especificado pelo usuário?
- **Custo do caminho**: Ele depende do valor da moeda, do tempo de espera, horário do voo, procedimentos de imigração e de atendimento ao cliente, qualidade do assento, horário do dia, tipo de avião, milhagem acumulada etc.

Os sistemas comerciais de informações para viagens utilizam uma formulação de problema desse tipo, com muitas complicações adicionais para manipular as estruturas bizantinas de tarifas que as empresas aéreas impõem. Porém, qualquer viajante experiente sabe que nem toda viagem aérea transcorre de acordo com os planos. Um sistema realmente bom deve incluir planos de contingência — como reservas de emergência em voos alternativos — até o ponto em que esses planos possam ser justificados pelo custo e pela probabilidade de fracasso do plano original.

Os **problemas de roteiro de viagem** estão diretamente relacionados aos problemas de roteamento, mas apresentam uma importante diferença. Por exemplo, considere o problema: "Visitar cada cidade da Figura 3.2 pelo menos uma vez, começando e terminando em Bucareste." Como ocorre com o roteamento, as ações correspondem a viagens entre cidades adjacentes. Porém, o espaço de estados é bastante diferente. Cada estado deve incluir não apenas a posição atual, mas também o *conjunto de cidades que o agente visitou*. Assim, o estado inicial seria "*Em*(*Bucareste*), *Visitado*({*Bucareste*})", um estado intermediário típico seria "*Em*(*Vaslui*), *Visitado*({*Bucareste*, *Urziceni*, *Vaslui*})", e o teste de objetivo verificaria se o agente está em Bucareste e se todas as 20 cidades foram visitadas.

O **problema do caixeiro-viajante** (TSP) é um problema de roteiro de viagem em que cada cidade deve ser visitada exatamente uma vez. O objetivo é encontrar o percurso *mais curto*. O problema é conhecido por ser NP-difícil, mas um grande esforço tem sido empregado para melhorar os recursos de algoritmos de TSP. Além de planejar viagens para caixeiros-viajantes, esses algoritmos são usados para tarefas como planejar movimentos de máquinas automáticas para perfuração de placas de circuitos e de máquinas industriais em fábricas.

Um problema de **leiaute de circuitos eletrônicos VLSI** exige o posicionamento de milhões de componentes e conexões em um chip para minimizar a área, minimizar retardos de circuitos, minimizar capacitâncias de fuga e maximizar o rendimento industrial. O problema de leiaute vem depois da fase do projeto lógico, e normalmente se divide em duas partes: **leiaute de células** e **roteamento de canais**. No leiaute de células, os componentes primitivos do circuito são agrupados em células, cada uma das quais executa alguma função conhecida. Cada célula tem uma área ocupada fixa (tamanho e forma) e exige um certo número de conexões para cada uma das outras células. O

objetivo é dispor as células no chip de tal forma que elas não se sobreponham e exista espaço para que os fios de conexão sejam colocados entre as células. O roteamento de canais encontra uma rota específica para cada fio passando pelos espaços vazios entre as células. Esses problemas de busca são extremamente complexos, mas sem dúvida vale a pena resolvê-los. Mais adiante neste capítulo, apresentaremos alguns algoritmos capazes de solucioná-los.

A **navegação de robôs** é uma generalização do problema de roteamento descrito anteriormente. Em vez de seguir um conjunto discreto de rotas, um robô pode se mover em um espaço contínuo com (em princípio) um conjunto infinito de ações e estados possíveis. No caso de um robô em movimento circular sobre uma superfície plana, o espaço é essencialmente bidimensional. Quando o robô tem braços e pernas ou rodas que também devem ser controlados, o espaço de busca passa a ter várias dimensões. São exigidas técnicas avançadas só para tornar finito o espaço de busca. Examinaremos alguns desses métodos no Capítulo 25. Além da complexidade do problema, robôs reais também devem lidar com erros nas leituras de seus sensores e nos controles do motor.

A **sequência automática de montagem** de objetos complexos por um robô foi demonstrada primeiramente por FREDDY (Michie, 1972). Desde então, o progresso tem sido lento mas seguro, até chegar ao ponto em que a montagem de objetos complexos como motores elétricos se torna economicamente viável. Em problemas de montagem, o objetivo é encontrar uma ordem na qual devem ser montadas as peças de algum objeto. Se for escolhida a ordem errada, não haverá como acrescentar alguma peça mais adiante na sequência sem desfazer uma parte do trabalho já realizado. A verificação da viabilidade de um passo na sequência é um problema geométrico difícil de busca, intimamente relacionado à navegação de robôs. Desse modo, a geração de ações válidas é a parte dispendiosa da sequência de montagem. Qualquer algoritmo prático deve evitar explorar mais do que uma fração minúscula desse espaço de estados. Outro problema de montagem importante é o **projeto de proteínas**, em que o objetivo é encontrar uma sequência de aminoácidos que serão incorporados em uma proteína tridimensional com as propriedades adequadas para curar alguma doença.

3.3 EM BUSCA DE SOLUÇÕES

Depois de formular alguns problemas, precisamos resolvê-los. Uma solução é uma sequência de ações, de modo que os algoritmos de busca consideram várias sequências de ações possíveis. As sequências de ações possíveis que começam a partir do estado inicial formam uma **árvore de busca** com o estado inicial na raiz; os ramos são as ações, e os **nós** correspondem aos estados no espaço de estados do problema. A Figura 3.6 apresenta os primeiros passos no crescimento da árvore de busca para encontrar uma rota de Arad para Bucareste. O nó raiz da árvore corresponde ao estado inicial, *Em(Arad)*. O primeiro passo é testar se esse é um estado objetivo (é claro que não é, mas é importante verificar isso, a fim de podermos resolver problemas complicados como "partindo de Arad, chegar a Arad"). Então é necessário considerar a escolha de diversas ações. Isso é feito pela **expansão** do estado atual, ou seja, aplicando cada ação válida no estado atual, **gerando** assim um novo conjunto de estados. Nesse caso, adicionaremos três novos ramos a partir do **nó pai** *Em*(*Arad)* conduzindo a três novos **nós filhos**: *Em*(*Sibiu*), *Em*(*Timisoara*) e *Em*(*Zerind*). Agora temos de escolher qual dessas três possibilidades merece consideração adicional.

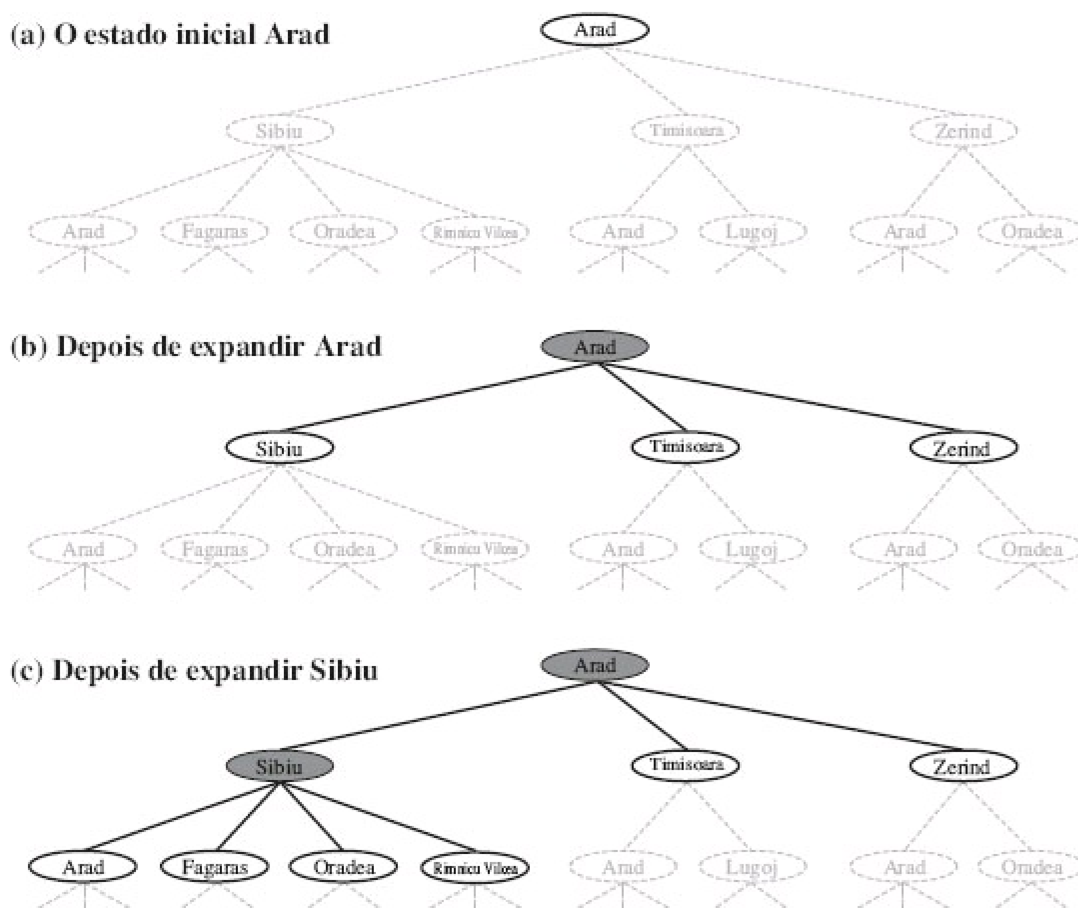


Figura 3.6 Árvores de busca parciais para localização de uma rota desde Arad até Bucareste. Nós que foram expandidos estão sombreados; nós que foram gerados mas ainda não foram expandidos têm contorno em negrito; nós que ainda não foram gerados são mostrados em linhas leves tracejadas.

Essa é a essência da busca — seguir uma opção agora e deixar as outras reservadas para mais tarde, no caso da primeira escolha não levar a uma solução. Vamos supor que escolhemos primeiro Sibiu. Verificamos se ela é um estado objetivo (não é) e depois o expandimos para obter *Em*(*Arad*), *Em*(*Fagaras*) , *Em*(*Oradea*) e *Em*(*RimnicuVilcea*). Portanto, podemos escolher qualquer dessas quatro opções ou então voltar e escolher Timisoara ou Zerind. Cada um desses seis nós é um **nó folha**, ou seja, um nó sem filhos na árvore. O conjunto de todos os nós folhas disponíveis para expansão em um dado ponto é chamado de **borda** (muitos autores chamam de **lista aberta**, que é tanto gratificante menos significativo quanto menos preciso, pois outras estruturas de dados são mais adequadas do que uma lista). Na Figura 3.6, a borda de cada árvore é constituída por aqueles nós com contornos em negrito.

O processo de expansão dos nós na borda continua até que uma solução seja encontrada ou não existam mais estados a expandir. O algoritmo geral BUSCA-EM-ÁRVORE é mostrado informalmente na Figura 3.7. Todos os algoritmos de busca compartilham essa estrutura básica; eles variam fundamentalmente de acordo com a forma escolhida para o próximo estado a expandir — a chamada **estratégia de busca**.

função BUSCA-EM-ÁRVORE(*problema*) **retorna** uma solução ou falha
  inicializar a borda utilizando o estado inicial do *problema*
  repita
    se *borda vazia* **então retornar** falha

```
    escolher um nó folha e o remover da borda
    se o nó contém um estado objetivo então retornar a solução correpondente
    expandir o nó escolhido, adicionando os nós resultantes à borda

função BUSCA-EM-GRAFO(problema) retorna uma solução ou falha
  inicializar a borda utilizando o estado inicial do problema
  inicializar o conjunto explorado tornando-o vazio
  repita
    se borda vazia então retornar falha
    escolher um nó folha e o remover da borda
    se o nó contiver um estado objetivo então retornar a solução correpondente
    adicionar o nó ao conjunto explorado
    expandir o nó escolhido, adicionando os nós resultantes à borda
      apenas se não estiver na borda ou no conjunto explorado
```

Figura 3.7 Descrição informal do algoritmo geral de busca em árvore e busca em grafo. As partes da BUSCA-EM-GRAFO marcadas em negrito e itálico são as adições necessárias para tratar estados repetidos.

O leitor atento notará algo peculiar sobre a árvore de busca mostrada na Figura 3.6: inclui o caminho de Arad para Sibiu e de volta para Arad novamente! Dizemos que *Em(Arad)* é um **estado repetido** na árvore de busca, gerado, nesse caso, por um **caminho em laço**. A consideração de tais caminhos em laço significa que a árvore de busca completa para a Romênia é *infinita* porque não há limite de quantas vezes se pode percorrer um laço. Por outro lado, o espaço de estados — o mapa mostrado na Figura 3.2 — tem apenas 20 estados. Como discutimos na Seção 3.4, os laços podem fazer com que certos algoritmos falhem, tornando insolúveis os problemas que seriam solúveis. Felizmente, não é necessário considerar caminhos em laço. Podemos contar mais com a intuição: porque custos de caminho são aditivos e os custos de passo são não negativos, um caminho em laço para qualquer estado dado nunca será melhor do que o mesmo caminho com o laço removido.

Caminhos em laço são um caso especial do conceito mais geral de **caminhos redundantes**, que existem sempre que houver mais de uma maneira para ir de um estado a outro. Considere os caminhos Arad-Sibiu (140 km de comprimento) e Arad-Zerind-Oradea-Sibiu (297 km de comprimento). Obviamente, o segundo caminho é redundante — é apenas uma forma pior de chegar ao mesmo estado. Se você estiver preocupado em alcançar o objetivo, nunca haverá qualquer razão para manter mais de um caminho para qualquer estado dado porque qualquer estado objetivo que pode ser acessado através da extensão de um caminho também será acessível através da extensão do outro.

Em alguns casos, é possível definir o problema em si de modo a eliminar os caminhos redundantes. Por exemplo, se formularmos o problema das oito rainhas de modo que uma rainha possa ser colocada em qualquer coluna, cada estado com n rainhas poderá ser alcançado por $n!$ caminhos diferentes; mas, se reformularmos o problema de modo que cada nova rainha seja colocada na coluna à esquerda vazia, então cada estado poderá ser alcançado apenas através de um caminho.

Em outros casos, caminhos redundantes são inevitáveis. Inclui todos os problemas nos quais as ações são reversíveis, como problemas de roteamento e quebra-cabeças de blocos deslizantes. A busca de rota em uma **grade retangular** (como a utilizada mais adiante na Figura 3.9) é um exemplo particularmente importante em jogos de computador. Em tal grade, cada estado tem quatro sucessores; assim, uma árvore de busca de profundidade d, que inclui estados repetidos tem folhas 4^d, mas há apenas cerca de $2d^2$ estados distintos dentro de d passos de determinado estado. Para d = 20, significa cerca de um trilhão de nós, mas apenas cerca de 800 estados distintos. Assim, seguir caminhos redundantes pode fazer com que um problema tratável se torne intratável. Isso é verdadeiro mesmo para os algoritmos que sabem como evitar laços infinitos.

Como diz o ditado, *os algoritmos que se esquecem de sua história estão fadados a repeti-la.* A maneira de evitar a exploração de caminhos redundantes é lembrar por onde passou. Para fazer isso, aumentaremos o algoritmo da BUSCA-EM-ÁRVORE com uma estrutura de dados chamada de **conjunto explorado** (também conhecida como **lista fechada**), que lembra de todo o nó que foi expandido. Nós que foram gerados recentemente que casam com nós anteriormente gerados — os que estão no conjunto explorado ou na borda — podem ser descartados em vez de adicionados à borda. O novo algoritmo, chamado BUSCA-EM-GRAFO, é apresentado informalmente na Figura 3.7. Os algoritmos específicos deste capítulo são extraídos desse projeto geral.

Claramente, a árvore de busca construída pelo algoritmo de BUSCA-EM-GRAFO, contém no máximo uma cópia de cada estado; assim, podemos imaginá-la como uma árvore que cresce diretamente sobre o grafo do espaço de estados, como mostrado na Figura 3.8. O algoritmo tem outra propriedade interessante: a borda **separa** o grafo do espaço de estados em região explorada e inexplorada, de modo que cada caminho do estado inicial para o estado inexplorado tem que passar por um estado da borda (se isso parece óbvio, tente agora o Exercício 3.13). Essa propriedade é ilustrada na Figura 3.9. Como cada passo move um estado da borda para a região explorada, enquanto alguns estados da região inexplorada movem-se para a borda, vemos que o algoritmo está analisando *sistematicamente* os estados no espaço de estados, um por um, até encontrar uma solução.

3.3.1 Infraestrutura para algoritmos de busca

Algoritmos de busca exigem uma estrutura de dados para manter o controle da árvore de busca que está sendo construída. Para cada nó *n* da árvore, temos uma estrutura que contém quatro componentes:

- *n*.ESTADO: o estado no espaço de estado a que o nó corresponde;
- *n*.PAI: o nó na árvore de busca que gerou esse nó;
- *n*.AÇÃO: a ação que foi aplicada ao pai para gerar o nó;
- *n*.CUSTO-DO-CAMINHO: o custo, tradicionalmente denotado por $g(n)$, do caminho do estado inicial até o nó, indicado pelos ponteiros para os pais.

Dados os componentes de um nó pai, é fácil verificar como calcular os componentes necessários para um nó filho. A função de NÓ-FILHO toma um nó pai e uma ação e devolve o nó filho resultante:

função NÓ-FILHO(*problema, pai, ação*) **retorna** um nó
retorna um nó com
ESTADO = *problema*.RESULTADO(*pai*.ESTADO, *ação*),
PAI = pai, AÇÃO = ação,
CUSTO-DE-CAMINHO = *pai*.CUSTO-DE-CAMINHO + *problema*.CUSTO-DO-PASSO(*pai*.ESTADO, *ação*)

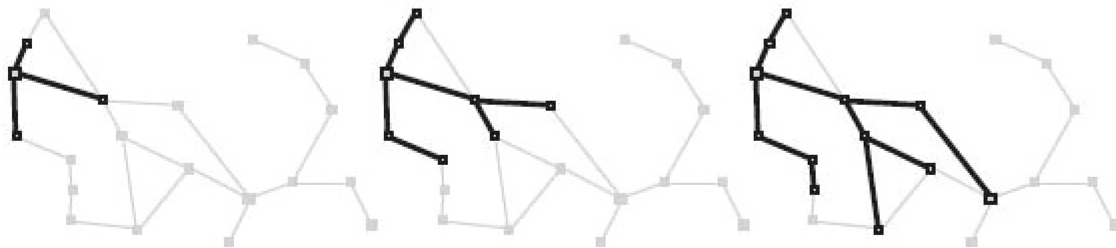


Figura 3.8 Uma sequência de árvores de busca gerada pela busca em grafo no problema da Romênia, da Figura 3.2. Em cada estágio, estendemos cada caminho em um passo. Repare que, na terceira fase, a cidade mais ao norte (Oradea) tornou-se um beco sem saída: os seus sucessores já foram explorados através de outros caminhos.

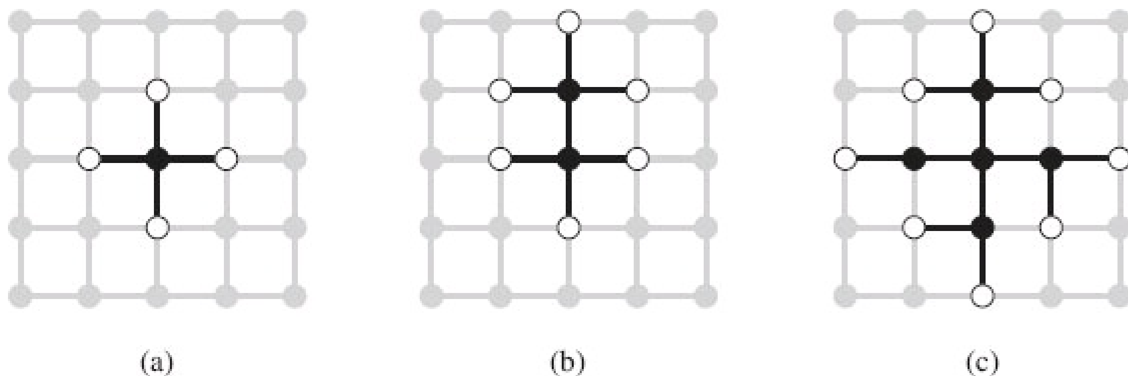


Figura 3.9 A propriedade de separação da BUSCA-EM-GRAFO foi ilustrada no problema da grade retangular. A borda (nós brancos) sempre separa a região explorada do espaço de estados (nós negros) da região inexplorada (nós cinzas). Em (a), apenas a raiz foi expandida. Em (b), um nó folha foi expandido. Em (c), os sucessores restantes da raiz foram expandidos no sentido horário.

A estrutura de dados do nó está representada na Figura 3.10. Observe como os ponteiros do PAI encadeiam os nós juntos em uma estrutura de árvore. Esses ponteiros também permitem que o caminho da solução seja extraído quando é encontrado um nó objetivo; utiliza-se a função SOLUÇÃO para devolver a sequência de ações obtidas seguindo os ponteiros do pai de volta para a raiz.

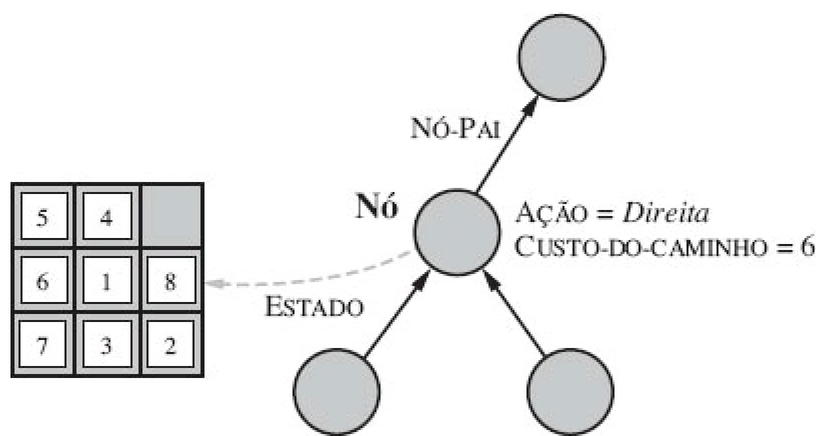


Figura 3.10 Nós são as estruturas de dados a partir das quais a árvore de busca é construída. Cada nó tem um pai, um estado e diversos campos de anotação. As setas (ponteiros) apontam do filho para o pai.

Até agora, não fomos muito cuidadosos na distinção entre nós e estados, mas ao escrever algoritmos detalhados é importante fazer essa distinção. Um nó é uma anotação da estrutura de dados usada para representar a árvore de busca. Um estado corresponde a uma configuração do mundo. Assim, os nós estão em caminhos particulares, tal como definido pelos ponteiros PAI, enquanto os estados não estão. Além disso, dois nós diferentes podem conter o mesmo estado do mundo se esse estado for gerado através de dois caminhos de busca diferentes.

Agora, que temos os nós, precisamos de um lugar para colocá-los. A borda precisa ser armazenada de tal forma que o algoritmo de busca possa facilmente escolher o próximo nó para expandir de acordo com sua estratégia preferida. A estrutura de dados apropriada para isso é uma **fila**. As operações sobre uma fila são:

- VAZIA?(*fila*) devolve verdadeiro somente se não existir mais nenhum elemento na fila.
- POP(*fila*) remove o primeiro elemento da fila e o devolve.
- INSERIR(*elemento*, *fila*) insere um elemento na fila e devolve a fila resultante.

As filas são caracterizadas pela *ordem* em que armazenam os nós inseridos. Três variantes mais comuns são: o primeiro a entrar na fila é o primeiro a sair ou a **fila FIFO** (first in, first out), que dispara o elemento *mais antigo* da fila; o último a entrar na fila é o último a sair ou **fila LIFO** (last in, last out) (também conhecida como **pilha**), que dispara o elemento *mais recente* da fila; e a **fila de prioridade**, que dispara o elemento da fila com a maior prioridade de acordo com alguma função de ordenação.

O conjunto explorado pode ser implementado com uma tabela hash para permitir um controle eficaz de estados repetidos. Com uma boa implementação, a inserção e a busca podem ser realizadas em tempos aproximadamente constantes, não importando quantos estados estiverem armazenados. Deve-se ter o cuidado de implementar a tabela hash com a correta noção de igualdade entre estados. Por exemplo, no problema do caixeiro-viajante, é necessário que a tabela hash saiba que o conjunto de cidades visitadas {Bucareste, Urziceni, Vaslui} é o mesmo que {Urziceni, Vaslui, Bucharest}. Às vezes isso pode ser alcançado mais facilmente, forçando que as estruturas de dados para os estados estejam em alguma **forma canônica,** isto é, estados logicamente equivalentes deverão mapear a mesma estrutura de dados. No caso de estados descritos por conjuntos, por exemplo, uma

representação de um vetor de bits ou uma lista ordenada sem repetições seriam canônicos, ao passo que uma lista não ordenada, não seria.

3.3.2 Medição de desempenho de resolução de problemas

Antes de entrar no projeto de algoritmos de busca específicos, precisamos considerar os critérios que podem ser usados para se fazer uma escolha entre eles. Podemos avaliar o desempenho do algoritmo em quatro aspectos:

- **Completeza**: O algoritmo oferece a garantia de encontrar uma solução quando ela existir?
- **Otimização:** A estratégia encontra a solução ótima, como definido na página 68?
- **Complexidade de tempo:** Quanto tempo ele leva para encontrar uma solução?
- **Complexidade de espaço:** Quanta memória é necessária para executar a busca?

A complexidade de tempo e a complexidade de espaço de memória são sempre consideradas em relação a alguma medida da dificuldade do problema. Em ciência da computação teórica, a medida típica é o tamanho do grafo do espaço de estados, $|V| + |E|$, onde V é o conjunto de vértices (nós) do grafo e E é o conjunto de arestas (arcos). Isso é apropriado quando o grafo for uma estrutura de dados explícita que é a entrada para o programa de busca (o mapa da Romênia é um exemplo dessa estrutura). Em IA, o grafo é sempre representado *implicitamente* pelo estado inicial, ações, modelo de transição, com frequência infinito. Por essas razões, a complexidade é expressa em termos de três quantidades: b, o **fator de ramificação** ou número máximo de sucessores de qualquer nó; d, a **profundidade** do nó objetivo menos profundo (ou seja, o número de passos ao longo do caminho da raiz até o estado objetivo mais próximo); e m, o comprimento máximo de qualquer caminho no espaço de estados. Com frequência, o tempo é medido em termos do número de nós gerados durante a busca, e o espaço é medido em termos do número máximo de nós armazenados na memória. Na maior parte, descreveremos a complexidade de tempo e espaço para a busca em uma árvore. Para um grafo, a resposta depende do quão “redundante” são os caminhos no espaço de estados.

Para avaliar a efetividade de um algoritmo de busca, podemos considerar apenas **o custo de busca** — que, em geral, depende da complexidade de tempo, mas também pode incluir um termo para uso da memória — ou podemos usar o **custo total**, que combina o custo de busca e o custo de caminho da solução encontrada. Para o problema de localizar uma rota desde Arad até Bucareste, o custo de busca é o período de tempo exigido pela busca, e o custo de solução é o comprimento total do caminho em quilômetros. Desse modo, para calcular o custo total, temos de somar quilômetros e milissegundos. Não existe nenhuma “taxa de conversão oficial” entre essas duas unidades de medida, mas, nesse caso, talvez fosse razoável converter quilômetros em milissegundos usando uma estimativa da velocidade média do carro (porque o tempo é o que importa ao agente). Isso permite ao agente encontrar um ponto de equilíbrio ótimo, no qual se torna contraproducente realizar computação adicional para encontrar um caminho mais curto. O problema mais geral de compensação entre diferentes recursos será examinado no Capítulo 16.

3.4 ESTRATÉGIAS DE BUSCA SEM INFORMAÇÃO

Esta seção focaliza diversas estratégias de busca reunidas sob o título de **busca sem informação** (também chamada **busca cega**). A expressão significa que elas não têm nenhuma informação adicional sobre estados, além daquelas fornecidas na definição do problema. Tudo o que elas podem fazer é gerar sucessores e distinguir um estado objetivo de um estado não objetivo. Todas as estratégias de busca se distinguem pela *ordem* em que os nós são expandidos. As estratégias que sabem se um estado não objetivo é "mais promissor" que outro são chamadas estratégias de **busca informada** ou **busca heurística**; elas serão estudadas na Seção 3.5.

3.4.1 Busca em largura

A **busca em largura** (BrFS – Breadth-first search) é uma estratégia simples em que o nó raiz é expandido primeiro, em seguida todos os sucessores do nó raiz são expandidos, depois os sucessores *desses nós*, e assim por diante. Em geral, todos os nós em dada profundidade na árvore de busca são expandidos, antes que todos os nós no nível seguinte sejam expandidos.

A busca em largura é uma instância do algoritmo de busca em grafo (Figura 3.7), em que o nó mais raso não expandido é escolhido para expansão. Isso é conseguido simplesmente utilizando uma fila FIFO para a borda. Assim, novos nós (que são sempre mais profundos do que seus pais) vão para o fim da fila, e nós antigos, que são mais rasos que os novos, são expandidos primeiro. Há um ligeiro refinamento no algoritmo genérico de busca em grafos, pois o teste de objetivo é aplicado a cada nó quando é *gerado* e não quando é selecionado para expansão. Essa decisão será explicada a seguir, quando discutirmos a complexidade do tempo. Observe também que o algoritmo, que segue o modelo geral para a busca em grafos, descarta qualquer caminho novo para um estado já na borda ou no conjunto explorado; é fácil verificar que tal caminho deverá pelo menos ser tão profundo quanto aquele já encontrado. Assim, a busca em largura sempre terá o caminho mais raso para todo o nó na borda.

O pseudocódigo é apresentado na Figura 3.11. A Figura 3.12 mostra o progresso da busca em uma árvore binária simples.

```
função BUSCA-EM-LARGURA(problema) retorna uma solução ou falha
  nó ← um nó com ESTADO = problema.ESTADO-INICIAL, CUSTO-DE-CAMINHO = 0
  se problema.TESTE-DE-OBJETIVO(nó.ESTADO) senão retorne SOLUÇÃO(nó),
  borda ← uma fila FIFO com nó como elemento único
  explorado ← conjunto vazio
  repita
    se VAZIO?(borda), então retorne falha
    nó ← POP(borda) / * escolhe o nó mais raso na borda */
    adicione nó.ESTADO para explorado
    para cada ação em problema.AÇÕES(nó.ESTADO) faça
      filho ← NÓ-FILHO(problema, nó, ação),
```

```
        se (filho.ESTADO)não está em explorado ou borda então
           se problema.TESTE-DE-OBJETIVO(filho.ESTADO) então retorne SOLUÇÃO(filho)
           borda ← INSIRA(filho, borda)
```

Figura 3.11 Busca em largura em um grafo.

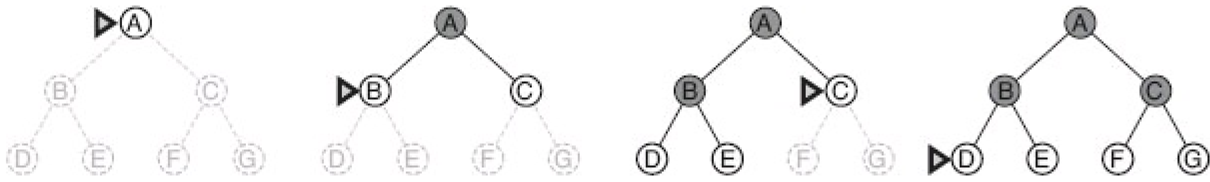


Figura 3.12 Busca em largura em uma árvore binária simples. Em cada fase, o próximo nó a ser expandido é indicado por um marcador.

Como avaliar a busca em largura de acordo com os quatro critérios da seção anterior? Podemos facilmente verificar que ela é c*ompleta* — se o nó objetivo mais raso estiver em alguma profundidade finita *d*, a busca em largura acabará por encontrá-lo depois de gerar todos os nós mais rasos (desde que o fator de ramificação *b* seja finito). Observe que, logo que um nó objetivo é gerado, sabemos que é o nó objetivo mais raso porque todos os nós mais rasos já devem ter sido gerados e falhado no teste de objetivo. Agora, o nó objetivo *mais raso* não é necessariamente o *ótimo*; tecnicamente, a busca em largura é ideal se o custo do caminho for uma função não decrescente da profundidade do nó. O mais comum desse cenário é que todas as ações têm o mesmo custo.

Até aqui, o que se falou sobre busca em largura tem sido boas notícias. As notícias sobre o tempo e o espaço não são tão boas. Imagine a busca em uma árvore uniforme onde cada estado tenha *b* sucessores. A raiz da árvore de busca gera *b* nós no primeiro nível, cada um dos quais gera *b* outros nós, totalizando b^2 no segundo nível. Cada um *desses* outros nós gera *b* outros nós, totalizando b^3 nós no terceiro nível, e assim por diante. Agora, suponha que a solução esteja na profundidade *d*. No pior caso, é o último nó gerado naquele nível. Então, o número total de nós gerados é:

$$b + b^2 + b^3 + \ldots + b^d = O(b^d).$$

(Se o algoritmo fosse aplicar o teste de objetivo para nós ao serem selecionados para a expansão, em vez de ao serem gerados, toda a camada de nós na profundidade *d* seria expandida antes que o objetivo fosse detectado e a complexidade de tempo seria $O(b^{d+1})$.).

Quanto à complexidade do espaço: para qualquer tipo de busca em grafos, que armazena todos os nós expandidos no conjunto *explorado*, a complexidade do espaço está sempre dentro de um fator de *b* da complexidade do tempo. Em particular, para busca em largura em grafos, cada nó gerado permanecerá na memória. Haverá $O(b^{d-1})$ nós no conjunto explorado e $O(b^d)$ nós na borda; assim, a complexidade de espaço será $O(b^d)$, ou seja, será dominada pelo tamanho da borda. Mudar para uma busca em árvore não iria poupar muito espaço e, em um espaço de estados com muitos caminhos redundantes, a mudança poderia custar grande parte do tempo.

Um limite de complexidade exponencial tal como $O(b^d)$, é assustador. A Figura 3.13 mostra por que, listando para vários valores de profundidade *d* da solução, o tempo e a memória necessários

para uma busca em largura com fator de ramificação b = 10. A tabela assume a geração de um milhão de nós por segundo e o requisito de 1.000 bytes de armazenamento para um nó. Muitos problemas podem ser ajustados a essas suposições (de um fator de 100 para mais ou para menos), quando executados em um computador pessoal moderno.

Profundidade	Nós	Tempo	Memória
2	110	0,11 milissegundo	107 kilobytes
4	11.110	11 milissegundos	10,6 megabytes
6	10^6	1,1 segundo	1 gigabyte
8	10^8	2 minutos	103 gigabytes
10	10^{10}	3 horas	10 terabytes
12	10^{12}	13 dias	1 petabyte
14	10^{14}	3,5 anos	99 petabytes
16	10^{16}	350 anos	10 exabytes

Figura 3.13 Requisitos de tempo e memória para a busca em largura. Os números mostrados assumem fator de ramificação b = 10; um milhão de nós/segundo; 1.000 bytes/nó.

Duas lições podem ser aprendidas a partir da Figura 3.13. Em primeiro lugar, *os requisitos de memória são um problema maior para a busca em largura do que o tempo de execução*. Deve-se esperar 13 dias pela solução de um problema importante com profundidade de busca 12, mas nenhum computador pessoal tem a memória principal da ordem de petabytes que ele exigiria. Felizmente, existem outras estratégias de busca que exigem menos memória.

A segunda lição é que o tempo ainda é um fator importante. Se seu problema tem uma solução na profundidade 16, então (dadas nossas suposições) ele demorará 350 anos para que a busca em largura a encontre (ou, na realidade, qualquer busca sem informação). Em geral, *os problemas de busca de complexidade exponencial não podem ser resolvidos por métodos sem informação, para qualquer instância, exceto as menores*.

3.4.2 Busca de custo uniforme

Quando todos os custos de passos forem iguais, a busca em largura será ótima porque sempre expande o nó *mais raso* não expandido. Através de uma simples extensão, podemos encontrar um algoritmo que é ótimo para qualquer função de custo do passo. Em vez de expandir o nó mais raso, a **busca de custo uniforme** expande o nó n com o *custo de caminho* $g(n)$ *mais baixo*. Isso é feito

através do armazenamento da borda como uma fila de prioridade ordenada por *g*. O algoritmo é mostrado na Figura 3.14.

```
função BUSCA-DE-CUSTO-UNIFORME(problema) retorna uma solução ou falha
   nó ← um nó com ESTADO = problema.ESTADO-INICIAL, CUSTO-DE-CAMINHO = 0
   borda ← fila de prioridade ordenada pelo CUSTO-DE-CAMINHO, com nó como elemento
 único
   explorado ← um conjunto vazio
   repita
      se VAZIO?(borda), então retornar falha
      nó ← POP(borda) / * escolhe o nó de menor custo na borda */
      se problema.TESTE-OBJETIVO(nó.ESTADO) então retornar SOLUÇÃO(nó)
      adicionar (nó.ESTADO) para explorado
      para cada ação em problema. AÇÕES(nó.ESTADO) faça
         filho ← NÓ-FILHO (problema, nó, ação)
         se (filho.ESTADO) não está na borda ou explorado então
            borda ← INSIRA (filho, borda)
         senão se (filho.ESTADO) está na borda com o maior CUSTO-DE-CAMINHO então
            substituir aquele nó borda por filho
```

Figura 3.14 Busca de custo uniforme em um grafo. O algoritmo é idêntico ao algoritmo geral de busca de grafo na Figura 3.7, exceto pelo uso de uma fila de prioridade e pela adição de uma verificação extra, caso um caminho mais curto para um estado de borda seja descoberto. A estrutura de dados para a *borda* deve permitir os testes eficientes de pertinência em conjunto, por isso deve combinar os recursos de uma fila de prioridade e de uma tabela hash.

Além da ordem da fila por custo do caminho, há duas outras diferenças significativas na busca em largura. A primeira é que o teste de objetivo é aplicado a um nó quando ele é *selecionado para a expansão* (como no algoritmo genérico da busca em grafos mostrado na Figura 3.7) e não quando é gerado pela primeira vez. A razão é que o primeiro nó objetivo que é *gerado* pode estar em um caminho abaixo do ótimo. A segunda diferença é que é adicionado um teste, caso seja encontrado um caminho melhor para um nó atualmente na borda.

Ambas as modificações entram em jogo no exemplo mostrado na Figura 3.15, onde o problema é ir de Sibiu para Bucareste. Os sucessores de Sibiu são Rimnicu Vilcea e Fagaras, com custos de 80 e 99, respectivamente. O nó de menor custo, Rimnicu Vilcea, será o próximo a ser expandido, acrescentando Pitesti com custo 80 + 97 = 177. O nó de menor custo é agora Fagaras, por isso será expandido, acrescentando Bucareste com custo de 99 + 211 = 310. Agora foi gerado um nó objetivo, mas a busca de custo uniforme se mantém, escolhendo Pitesti para expansão e adicionando um segundo caminho para Bucareste com um custo de 80 + 97 + 101 = 278. Em seguida o algoritmo verifica se esse novo caminho é melhor do que o antigo, isto é, de modo que o antigo seja descartado. Bucareste, agora com o custo *g* 278, será selecionada para a expansão e a solução será devolvida.

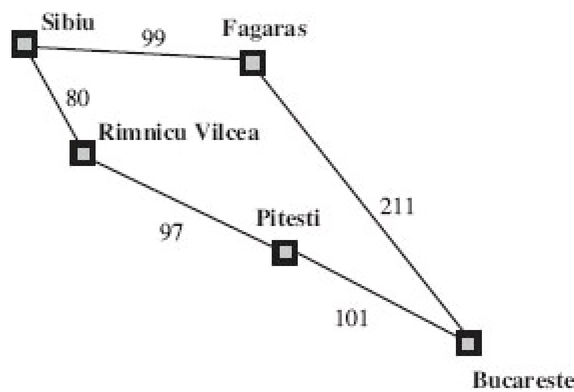


Figura 3.15 Parte do espaço de estados da Romênia, selecionada para ilustrar a busca de custo uniforme.

É fácil verificar que a busca de custo uniforme em geral é ótima. Em primeiro lugar, observamos que sempre que a busca de custo uniforme seleciona um nó n para expansão, o caminho ideal para esse nó foi encontrado (se esse não fosse o caso, haveria de ter outro nó na borda n' no caminho ótimo a partir do nó inicial até n, pela propriedade de separação de grafo da Figura 3.9; por definição, n' teria um custo g menor do que n e teria sido selecionado em primeiro lugar). Então, devido aos custos de passo serem não negativos, os caminhos nunca ficam menores à medida que os nós são adicionados. Esses dois fatos juntos implicam que a *busca de custo uniforme expande os nós na ordem de seu custo de caminho ótimo*. Assim, o primeiro nó objetivo que foi selecionado para expansão deverá ser a solução ótima.

A busca de custo uniforme não se importa com o *número* de passos que um caminho tem, mas apenas com o seu custo total. Por isso, ela ficará presa em um laço infinito se existir um caminho com sequência infinita de ações a custo zero — por exemplo, uma sequência de ações *NoOp*.[6] A completeza será garantida se o custo de cada passo exceder uma constante positiva pequena ϵ.

A busca de custo uniforme é orientada por custos de caminhos em vez de profundidades; assim, sua complexidade não pode ser caracterizada com facilidade em termos de b e d. Em vez disso, seja C^* o custo da solução ótima,[7] e suponha que toda ação custe pelo menos ϵ. Então, a complexidade de tempo e espaço do pior caso do algoritmo é $O(b^{1+\lfloor C^*/\epsilon \rfloor})$, que pode ser muito maior que b^d. Essa é a razão por que a busca de custo uniforme pode explorar grandes árvores de pequenos passos antes de explorar caminhos envolvendo passos grandes e talvez úteis. Quando todos os custos de passos forem iguais, $b^{1+\lfloor C^*/\epsilon \rfloor}$ será simplesmente b^{d+1}. Quando todos os custos de passo são iguais, a busca de custo uniforme é similar à busca em largura, exceto que a busca em largura para logo que gerar um objetivo, enquanto que a busca de custo uniforme examina todos os nós à profundidade do objetivo para verificar se algum deles tem custo mais baixo; portanto, a busca de custo uniforme tem mais trabalho, expandindo os nós à profundidade d, desnecessariamente.

3.4.3 Busca em profundidade

A **busca em profundidade** (DFS – Depth-first search) sempre expande o nó *mais profundo* na borda atual da árvore de busca. O progresso da busca é ilustrado na Figura 3.16. A busca prossegue imediatamente até o nível mais profundo da árvore de busca, onde os nós não têm sucessores. À

medida que esses nós são expandidos, eles são retirados da borda e, então, a busca “retorna” ao nó seguinte mais profundo que ainda tem sucessores inexplorados.

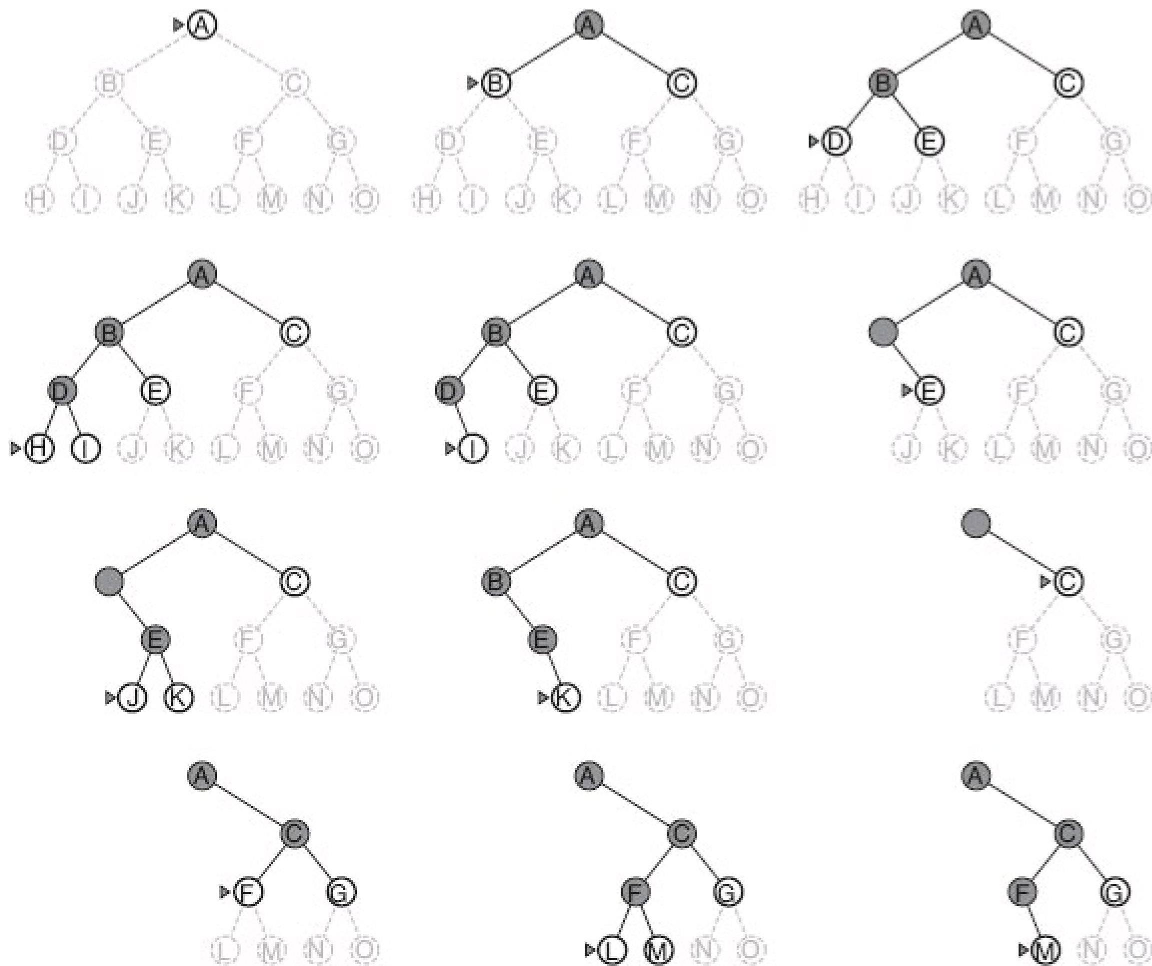


Figura 3.16 Busca em profundidade em uma árvore binária. A região inexplorada é mostrada em cinza-claro. Os nós explorados sem descendentes na borda são removidos da memória. Os nós na profundidade 3 não têm sucessores e *M* é o único nó objetivo.

O algoritmo de busca em profundidade é uma instância do algoritmo de busca em grafo da Figura 3.7; enquanto a busca em largura utiliza uma fila FIFO, a busca em profundidade utiliza uma fila LIFO. Uma fila LIFO significa que o nó gerado mais recentemente é escolhido para expansão. Deverá ser o nó mais profundo não expandido porque é mais profundo do que seu pai, que, por sua vez, era o nó não expandido mais profundo quando foi selecionado.

Como alternativa para a implementação do estilo da BUSCA EM GRAFOS, é comum implementar primeiro a busca em profundidade com uma função recursiva que chama a si mesma para cada um dos seus filhos por vez (a Figura 3.17 apresenta um algoritmo em profundidade recursivo incluindo um limite de profundidade).

função BUSCA-EM-PROFUNDIDADE-LIMITADA(problema, limite) **retorna** uma solução ou falha/corte
retornar BPL-RECURSIVA (CRIAR-NÓ(*problema,* ESTADO-INICIAL), *problema, limite*)
função BPL-RECURSIVA(*nó, problema, limite*) **retorna** uma solução ou falha/corte
  se *problema*. TESTAR-OBJETIVO (nó.ESTADO) então, **retorna** SOLUÇÃO (nó)
  se não se *limite* = 0 **então retorna** *corte*

```
senão
  corte_ocorreu? ← falso para cada ação no problema.AÇÕES(nó.ESTADO) faça
    filho ← NÓ-FILHO (problema, nó, ação)
    resultado ← BPL-RECURSIVA (criança, problema limite − 1)
    se resultado = corte  então corte_ocorreu? ← verdadeiro
    senão se resultado ≠ falha então retorna resultado
  se corte_ocorreu? então retorna corte senão retorna falha
```

Figura 3.17 Uma implementação recursiva da busca em árvore de profundidade limitada.

As propriedades da busca em profundidade dependem fortemente da versão utilizada da busca ser a busca em grafos ou a busca em árvore. A versão da busca em grafos, que evita estados repetidos e caminhos redundantes, é completa em espaços de estados finitos porque acabará por expandir cada nó. A versão da busca em árvore, por outro lado, *não* é completa — por exemplo, na Figura 3.6, o algoritmo seguirá o laço Arad-Sibiu-Arad-Sibiu para sempre. A busca em profundidade em árvore pode ser modificada, sem qualquer custo extra de memória, para verificar novos estados com relação aos nós do caminho da raiz até o nó atual; isso evita laços em espaço de estados finitos, mas não evita a proliferação de caminhos redundantes. Em espaço de estados infinitos, ambas as versões falham se um caminho não objetivo infinito for encontrado. Por exemplo, no problema 4 de Knuth, a busca em profundidade ficaria aplicando o operador fatorial para sempre.

Por motivos semelhantes, ambas as versões são não ótimas. Por exemplo, na Figura 3.16, a busca em profundidade vai explorar a subárvore esquerda toda, mesmo sendo o nó *C* um nó objetivo. Se o nó *J* também fosse um nó objetivo, a busca em profundidade iria devolvê-lo como solução em vez de *C*, que seria uma solução melhor; então, a busca em profundidade não é ótima.

A complexidade temporal da busca em profundidade em grafo é limitada pelo tamanho do espaço de estados (que certamente pode ser infinito). A busca em profundidade em árvore, por outro lado, poderá gerar todos os nós $O(b^m)$ na árvore de busca, onde m é a profundidade máxima de qualquer nó; isso pode ser muito maior do que o tamanho do espaço de estados. Observe que m em si pode ser muito maior do que d (profundidade da solução mais rasa) e é infinito se a árvore for ilimitada.

Até agora, a busca em profundidade parece não apresentar uma vantagem clara sobre a busca em largura, então por que incluí-la? O motivo é a complexidade espacial. Para uma busca em grafos, não há vantagem, mas uma busca em profundidade em árvore precisa armazenar apenas um único caminho da raiz até um nó folha, juntamente com os nós irmãos remanescentes não expandidos para cada nó no caminho. Uma vez que um nó é expandido, ele pode ser removido da memória, tão logo todos os seus descendentes tenham sido completamente explorados (veja a Figura 3.16). Para um espaço de estados com fator de ramificação b e profundidade máxima m, a busca em profundidade exige o armazenamento de apenas $O(bm)$ nós. Usando as mesmas suposições da Figura 3.13 e supondo que nós na mesma profundidade do nó objetivo não têm sucessores, verificamos que a busca em profundidade exigiria 156 kilobytes, em vez de 10 exabytes na profundidade $d = 16$, um espaço sete trilhões de vezes menor. Isso levou à adoção da busca em profundidade em árvore como o carro-chefe básico de muitas áreas da IA, incluindo a satisfação de restrição (Capítulo 6), a satisfatibilidade proposicional (Capítulo 7) e a programação lógica (Capítulo 9). Para o restante

desta seção, nos concentraremos principalmente na versão de busca em árvore da busca em profundidade.

Uma variante da busca em profundidade chamada **busca com retrocesso** utiliza ainda menos memória. (veja o Capítulo 6 para mais detalhes). No retrocesso, apenas um sucessor é gerado de cada vez, em lugar de todos os sucessores; cada nó parcialmente expandido memoriza o sucessor que deve gerar em seguida. Desse modo, é necessária apenas a memória $O(m)$, em vez de $O(bm)$. A busca com retrocesso permite ainda outro truque de economia de memória (e de economia de tempo): a ideia de gerar um sucessor pela *modificação* direta da descrição do estado atual, em vez de copiá-lo primeiro. Isso reduz os requisitos de memória a apenas uma descrição de estado e a $O(m)$ ações. Para que isso funcione, devemos ser capazes de desfazer cada modificação quando voltarmos para gerar o próximo sucessor. No caso de problemas com grandes descrições de estados, como a montagem robótica, essas técnicas são críticas para o sucesso.

3.4.4 Busca em profundidade limitada

O fracasso constrangedor da busca em profundidade em espaço de estados infinito pode ser atenuado pela busca em profundidade com um limite de profundidade predeterminado l. Isto é, nós na profundidade l são tratados como se não tivessem sucessores. Essa abordagem é chamada de **busca em profundidade limitada**. O limite de profundidade resolve o problema de caminhos infinitos. Infelizmente, ele também introduz uma fonte adicional de incompleteza, se escolhermos $l < d$, ou seja, o objetivo mais raso está além do limite de profundidade (isso não é improvável quando d é desconhecido). A busca em profundidade limitada também não será ótima se escolhermos $l > d$. Sua complexidade de tempo é $O(b^l)$ e sua complexidade de espaço é $O(bl)$. A busca em profundidade pode ser visualizada como um caso especial da busca em profundidade limitada com $l = \infty$.

Às vezes, limites de profundidade podem se basear no conhecimento que se tem do problema. Por exemplo, no mapa da Romênia há 20 cidades. Portanto, sabemos que, se existe uma solução, ela deve ter o comprimento 19 no caso mais longo, e então $l = 19$ é uma escolha possível. Porém, de fato, se estudássemos cuidadosamente o mapa, descobriríamos que qualquer cidade pode ser alcançada a partir de qualquer outra cidade em, no máximo, nove passos. Esse número, conhecido como **diâmetro** do espaço de estados, nos dá um limite de profundidade melhor, o que leva a uma busca em profundidade limitada mais eficiente. No entanto, na maioria dos problemas, não conhecemos um bom limite de profundidade antes de resolvermos o problema.

A busca em profundidade limitada pode ser implementada como uma modificação simples do algoritmo geral de busca em árvore ou em grafos. Alternativamente, pode ser implementada como um algorismo simples recursivo como mostrado na Figura 3.17. Observe que a busca em profundidade limitada pode terminar com dois tipos de falhas: o valor-padrão *falha* indica nenhuma solução; o valor *corte* indica nenhuma solução dentro do limite de profundidade.

3.4.5 Busca de aprofundamento iterativo

A **busca de aprofundamento iterativo** (IDS – Iterative Deepening Search) (ou busca em profundidade de aprofundamento iterativo) é uma estratégia geral, usada com frequência em combinação com a busca em profundidade em árvore, que encontra o melhor limite de profundidade. Ela faz isso aumentando gradualmente o limite — primeiro 0, depois 1, depois 2, e assim por diante até encontrar um objetivo. Isso ocorrerá quando o limite de profundidade alcançar d, a profundidade do nó objetivo mais raso. O algoritmo é mostrado na Figura 3.18. O aprofundamento iterativo combina os benefícios da busca em profundidade e da busca em largura. Como na busca em profundidade, seus requisitos de memória são muito modestos: $O(bd)$, para sermos precisos. Como na busca em largura, ele é completo quando o fator de ramificação é finito, e ótimo quando o custo de caminho é uma função não decrescente da profundidade do nó. A Figura 3.19 mostra quatro iterações da BUSCA-DE-APROFUNDAMENTO-ITERATIVO em uma árvore de busca binária, onde a solução é encontrada na quarta iteração.

função BUSCA-DE-APROFUNDAMENTO-ITERATIVO(*problema*) **retorna** uma solução ou falha
para profundidade = 0 **até** ∞ **faça**
resultado ← BUSCA-EM-PROFUNDIDADE-LIMITADA(*problema*, *profundidade*)
se *resultado* ≠ *corte* **então retornar** *resultado*

Figura 3.18 Algoritmo de busca de aprofundamento iterativo que aplica repetidamente a busca em profundidade limitada com limites crescentes. Ele termina quando uma solução é encontrada ou se a busca em profundidade limitada devolve *falha*, indicando que não existe nenhuma solução.

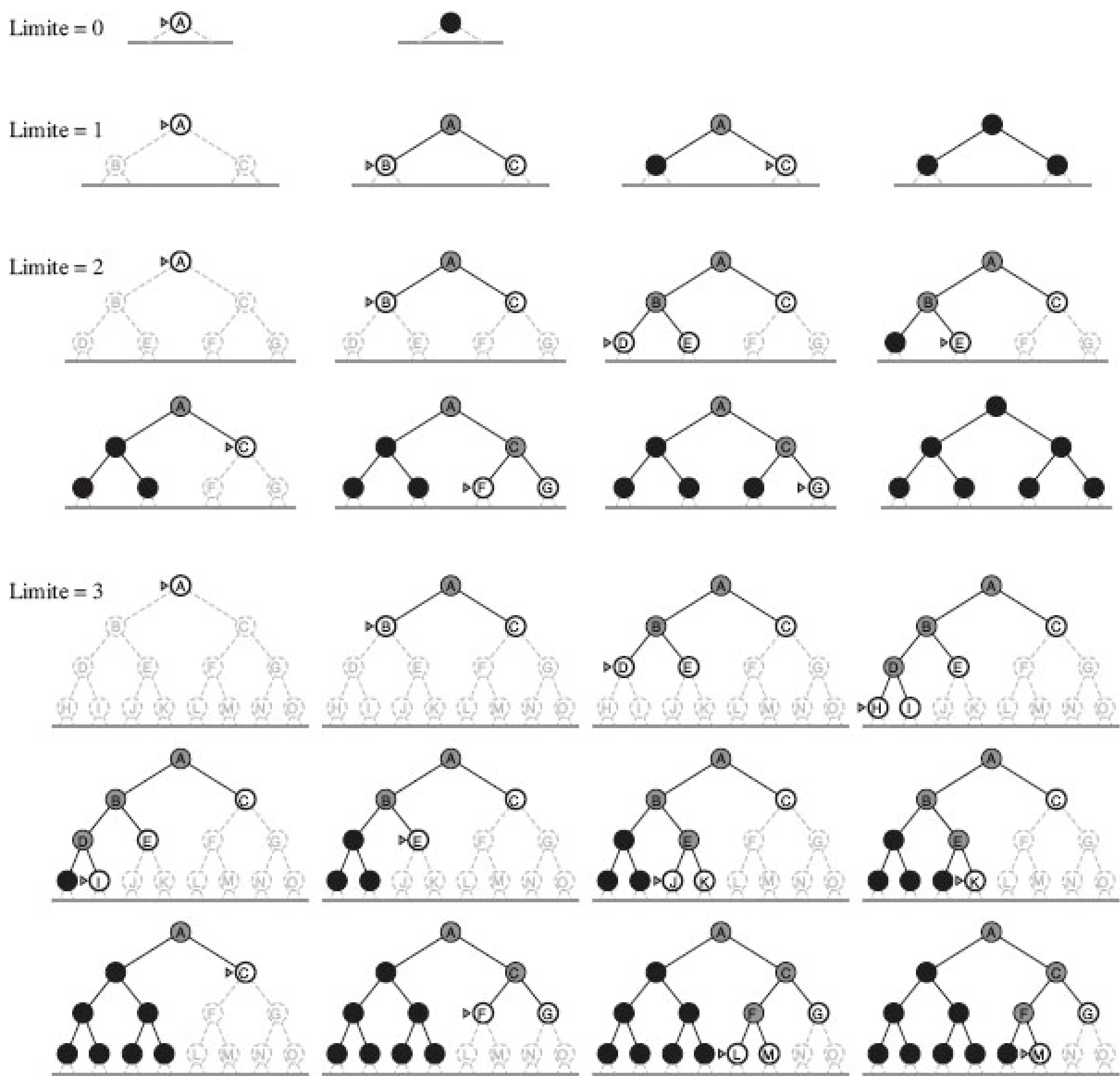


Figura 3.19 Quatro iterações de busca de aprofundamento iterativo em uma árvore binária.

A busca de aprofundamento iterativo pode parecer um desperdício porque os estados são gerados várias vezes. Na verdade, esse custo não é muito alto porque, em uma árvore de busca com o mesmo (ou quase o mesmo) fator de ramificação em cada nível, a maior parte dos nós estará no nível inferior e, assim, não importa muito se os níveis superiores são gerados várias vezes. Em uma busca de aprofundamento iterativo, os nós no nível inferior (profundidade d) são gerados uma vez, os do penúltimo nível inferior são gerados duas vezes, e assim por diante, até os filhos da raiz, que são gerados d vezes. Portanto, o número total de nós gerados é:

$$N(\text{IDS}) = (d)b + (d-1)b^2 + \ldots + (1)b^d,$$

o que dá uma complexidade de tempo igual a $O(b^d)$ — assintomaticamente, a mesma da busca em profundidade. Há um custo extra em gerar os níveis mais altos múltiplas vezes, mas não é grande. Por exemplo, se $b = 10$ e $d = 5$, os números são:

$$N(\text{IDS}) = 50 + 400 + 3.000 + 20.000 + 100.000 = 123.450$$
$$N(\text{BFS}) = 10 + 100 + 1.000 + 10.000 + 100.000 = 111.100.$$

Se você está realmente preocupado com a repetição da repetição, você pode usar uma abordagem híbrida que executa a busca em largura até que quase toda a memória disponível seja consumida, e então executar o aprofundamento iterativo de todos os nós na borda. *Em geral, o aprofundamento iterativo é o método de busca sem informação preferido quando o espaço de busca é grande e a profundidade da solução não é conhecida.*

A busca por aprofundamento iterativo é análoga à busca em largura, pelo fato de explorar uma camada completa de novos nós em cada iteração, antes de passar para a próxima camada. Parece que seria interessante desenvolver um algoritmo iterativo análogo à busca de custo uniforme, herdando as garantias de caráter ótimo do algoritmo anterior, ao mesmo tempo em que se reduz suas necessidades de memória. A ideia é usar limites crescentes de custo de caminho, em vez de limites crescentes de profundidade. O algoritmo resultante, chamado **busca de alongamento iterativo**, é explorado no Exercício 3.17. Infelizmente, o alongamento iterativo incorre em uma sobrecarga substancial, em comparação com a busca de custo uniforme.

3.4.6 Busca bidirecional

A ideia que rege a busca bidirecional é executar duas buscas simultâneas — uma direta, a partir do estado inicial, e a outra inversa, a partir do objetivo, esperando que as duas buscas se encontrem em um ponto intermediário (Figura 3.20). A motivação é que $b^{d/2} + b^{d/2}$ é muito menor que b^d ou, na figura, a área dos dois círculos pequenos é menor que a área do único círculo grande com centro no início e que chega até o objetivo.

Implementa-se a busca bidirecional substituindo o teste de objetivo por uma verificação para ver se as bordas das duas buscas se cruzam; se isso ocorre, foi encontrada uma solução. (É importante perceber que a primeira solução que foi encontrada pode não ser a ótima, mesmo que as duas buscas tenham sido em largura; haverá necessidade de uma busca adicional para se certificar de que não existe algum outro atalho através do espaço.) A verificação poderá ser realizada quando o nó for gerado ou selecionado para expansão e, com a tabela hash, terá um tempo constante. Por exemplo, se um problema tem solução à profundidade d = 6 e cada direção executa a busca em largura de um nó por vez, então, no pior caso, as duas buscas se encontram quando tiverem gerado todos os nós à profundidade 3. Para b = 10 isso é um total de 2.220 gerações de nós, comparado com 1.111.110 de uma busca em largura padrão. Assim, a complexidade de tempo da busca bidirecional usando a busca em largura nas duas direções é $O(b^{d/2})$. A complexidade do espaço é também $O(b^{d/2})$. Podemos reduzir isso por cerca da metade, se uma das duas buscas for feita por aprofundamento iterativo, mas, pelo menos uma das bordas deve ser mantida na memória para que possa ser feita a verificação do cruzamento. Esse requisito de espaço é a deficiência principal da busca bidirecional.

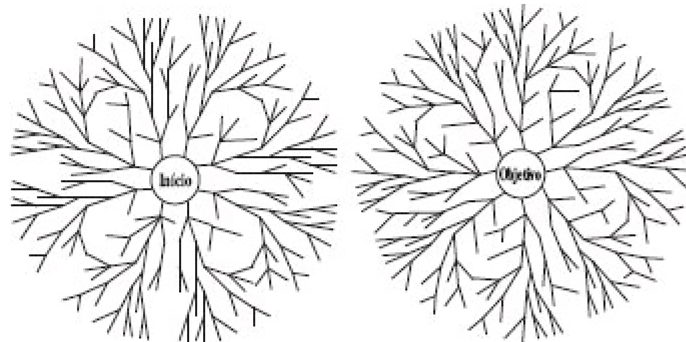


Figura 3.20 Visão esquemática de uma busca bidirecional prestes a ter sucesso quando uma ramificação a partir do nó inicial encontra uma ramificação a partir do nó objetivo.

A redução da complexidade de tempo torna a busca bidirecional atraente, mas como realizaremos a busca inversa? Isso não é tão fácil quanto parece. Sejam os **predecessores** de um estado x todos aqueles estados que têm x como sucessor. A busca bidirecional requer um método de cálculo dos predecessores. Quando todas as ações no espaço de estados forem reversíveis, os predecessores de x serão apenas seus sucessores. Outros casos podem exigir uma engenhosidade substancial.

Considere a questão do que queremos dizer com a palavra "objetivo" na frase "busca inversa a partir do objetivo". No caso do quebra-cabeça de oito peças e da localização de uma rota na Romênia, existe apenas um estado objetivo; assim, a busca inversa é muito semelhante à busca direta. Se houver vários estados objetivos *explicitamente listados* — por exemplo, os dois estados objetivo livres de sujeira da Figura 3.3 —, poderemos construir um novo estado objetivo fictício cujos predecessores imediatos serão todos os estados objetivo reais. Mas, se o objetivo for uma descrição abstrata, tal como o objetivo de que "nenhuma rainha ataque outra rainha" no problema n-rainhas, então é difícil usar a busca bidirecional.

3.4.7 Comparação entre estratégias de busca sem informação

A Figura 3.21 compara estratégias de busca em termos dos quatro critérios de avaliação definidos na Seção 3.3.2. Essa comparação refere-se às versões de busca em árvore. Para buscas em grafos, a principal diferença é que a busca em profundidade é completa para espaço de estados finitos e que as complexidades de espaço e de tempo estão limitadas pelo tamanho do espaço de estados.

Critério	Em largura	Custo uniforme	Em profundidade	Em profundidade limitada	Aprofundamento Iterativo	Bidirecion (se aplicável)
Completa?	Sim[a]	Sim[a,b]	Não	Não	Sim[a]	Sim[a,d]
Tempo	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Espaço	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Ótima?	Sim[c]	Sim	Não	Não	Sim[c]	Sim[c,d]

Figura 3.21 Avaliação de estratégias de busca em árvore. b é o fator de ramificação; d é a profundidade da solução mais rasa; m é a profundidade máxima da árvore de busca; l é o limite de

profundidade. As anotações sobrescritas são: [a] completa se b é finito; [b] completa se o custo do passo é $\geq \epsilon$ para ϵ positivo; [c] ótima se os custos dos passos são todos idênticos; [d] se ambos os sentidos utilizam busca em largura.

3.5 ESTRATÉGIA DE BUSCA INFORMADA (HEURÍSTICA)

Esta seção mostra como uma estratégia de **busca informada** — a que utiliza conhecimento de um problema específico além da definição do problema em si — pode encontrar soluções de forma mais eficiente do que uma estratégia de busca sem informação.

A abordagem geral que consideramos é chamada **busca de melhor escolha**. A busca de melhor escolha é uma instância do algoritmo geral da BUSCA-EM-ÁRVORE em que um nó é selecionado para a expansão com base em uma **função de avaliação**, $f(n)$. A função de avaliação é analisada como uma estimativa de custo, de modo que o nó com *a menor* avaliação será expandido primeiro. A implementação da busca em grafos de melhor escolha é idêntica à busca de custo uniforme (Figura 3.14), exceto pelo uso de f em vez de g para ordenar a fila de prioridade.

A escolha de f determina a estratégia de busca (por exemplo, como mostra o Exercício 3.21, a busca de melhor escolha em árvore inclui a busca em profundidade como caso especial). A maior parte dos algoritmos de melhor escolha inclui como componente de f uma função heurística, denotada por $h(n)$:

$h(n)$ = custo estimado do caminho de menor custo do estado do nó n para um estado objetivo.

(Note que $h(n)$ recebe um *nó* como entrada, mas, ao contrário de $g(n)$, depende apenas do *estado* naquele nó.) Por exemplo, na Romênia, pode-se estimar o custo do caminho de menor custo de Arad para Bucareste através da distância em linha reta de Arad para Bucareste.

Funções heurísticas são a forma mais comum como o conhecimento adicional do problema é transmitido ao algoritmo de busca. Estudaremos heurísticas mais profundamente na Seção 3.6. Por ora, consideramos as heurísticas como funções arbitrárias, não negativas, de problemas específicos, com uma restrição: se n for um nó objetivo, então $h(n) = 0$. O restante desta seção abrange duas maneiras de usar a informação heurística para orientar a busca.

3.5.1 Busca gulosa de melhor escolha

A **busca gulosa de melhor escolha**[8] tenta expandir o nó que está mais próximo do objetivo, com o fundamento de que isso pode conduzir a uma solução rapidamente. Assim, ela avalia os nós usando apenas a função heurística, ou seja, $f(n) = h(n)$.

Vamos ver como isso funciona para problemas de roteamento na Romênia; usaremos a heurística de **distância em linha reta** (DLR), que chamaremos de *hDLR*. Se o objetivo for Bucareste, precisaremos saber as distâncias em linha reta para Bucareste, apresentadas na Figura 3.22. Por exemplo, *hDLR* (*Em*(*Arad*)) = 366. Observe que os valores de *hDLR* não podem ser calculados da

descrição do problema em si. Além disso, é preciso certa experiência para saber que *hDLR* está correlacionado com as distâncias reais da estrada e, portanto, é uma heurística útil.

Arad	366	**Mehadia**	241
Bucareste	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figura 3.22 Valores de *hDLR* — distâncias em linha reta para Bucareste.

A Figura 3.23 mostra o andamento de uma busca gulosa de melhor escolha utilizando *hDLR* para encontrar um caminho de Arad para Bucareste. O primeiro nó a ser expandido a partir de Arad será Sibiu porque é mais perto de Bucareste do que Zerind ou Timisoara. O próximo nó a ser expandido será Fagaras porque é o mais próximo. Fagaras, por sua vez, vai gerar Bucareste, que é o objetivo. Para esse problema particular, a busca gulosa de melhor escolha utilizando *hDLR* encontra uma solução, sem nunca expandir um nó que não estiver no caminho da solução; portanto, seu custo de busca é mínimo. Não é ótimo, no entanto: o caminho via Sibiu e Fagaras para Bucareste é 32 quilômetros mais longo que o caminho através de Rimnicu Vilcea e Pitesti. Isso mostra por que o algoritmo é chamado de "ambicioso"; a cada passo ele tenta chegar o mais próximo do objetivo que puder.

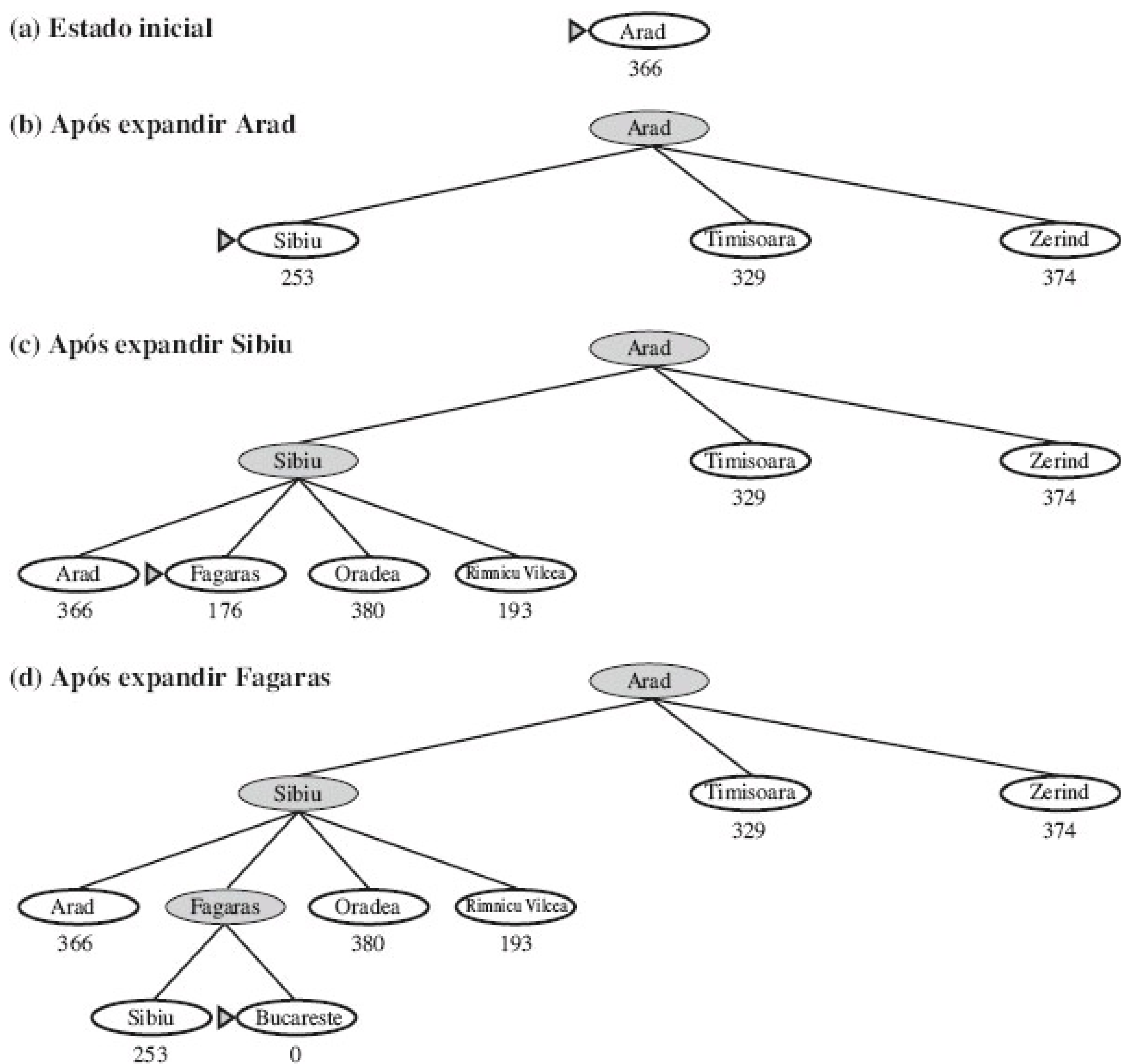


Figura 3.23 Etapas de uma busca gulosa de melhor escolha em árvore para Bucareste com a heurística de distância em linha reta *hDLR*. Os nós são rotulados com os seus valores *h*.

A busca gulosa de melhor escolha em árvore também é incompleta, mesmo em um espaço de estados finito, exatamente como a busca em profundidade. Considere o problema de ir de Iasi para Fagaras. A heurística sugere que Neamt seja expandida primeiro porque é mais próxima de Fagaras, mas isso é um beco sem saída. A solução é ir primeiro a Vaslui — um passo que, de acordo com a heurística, é na verdade mais longe do objetivo — e, em seguida, continuar para Urziceni, Bucareste e Fagaras. No entanto, o algoritmo nunca irá encontrar essa solução porque expandir Neamt coloca Iasi de volta na borda, Iasi está mais perto de Fagaras que Vaslui e, assim, Iasi será novamente expandida, levando a um laço infinito (a versão de busca em grafos *é* completa em espaços finitos, mas não infinitos). O pior caso de complexidade de tempo e de espaço para a versão em árvore é $O(b^m)$, onde m é a profundidade máxima do espaço de busca. Com uma boa função heurística, no entanto, a complexidade pode ser reduzida substancialmente. O montante da redução depende do problema particular e da qualidade da heurística.

3.5.2 Busca A*: minimização do custo total estimado da solução

A forma de solução mais amplamente conhecida da busca de melhor escolha é chamada de **busca A*** (pronuncia-se "busca A estrela"). Ela avalia os nós através da combinação de $g(n)$, o custo para alcançar o nó, e $h(n)$, o custo para ir do nó ao objetivo:

$$f(n) = g(n) + h(n).$$

Uma vez que $g(n)$ dá o custo do caminho desde o nó inicial até o nó n e $h(n)$ é o custo estimado do caminho de menor custo de n até o objetivo, teremos

$$f(n) = \text{custo estimado da solução de menor custo através de } n.$$

Assim, se estamos tentando encontrar a solução de menor custo, algo razoável para tentar em primeiro lugar, seria o nó com o menor valor de $g(n) + h(n)$. Acontece que essa estratégia é mais do que apenas razoável: desde que a função heurística $h(n)$ satisfaça certas condições, a busca A* será completa e ótima. O algoritmo é idêntico à BUSCA-DE-CUSTO-UNIFORME, exceto que A* usa $g + h$ em vez de g.

Condições para otimalidade: admissibilidade e consistência

A primeira condição requerida para otimalidade é que $h(n)$ seja uma **heurística admissível**. Uma heurística admissível é a que nunca superestima o custo de atingir o objetivo. Devido à $g(n)$ ser o custo real para atingir n ao longo do caminho atual, e $f(n) = g(n) + h(n)$, temos como consequência imediata que $f(n)$ nunca irá superestimar o verdadeiro custo de uma solução ao longo do caminho atual através de n.

Heurísticas admissíveis são otimistas por natureza porque imaginam que o custo de resolver o problema seja menor do que realmente é. Um exemplo óbvio de uma heurística admissível é a *hDLR* da distância em linha reta que usaremos para chegar a Bucareste. A distância em linha reta é admissível porque o caminho mais curto entre dois pontos quaisquer é uma linha reta, então a reta não pode ser uma superestimativa. Na Figura 3.24, mostramos a evolução de uma busca A* em árvore para Bucareste. Os valores de g são calculados a partir dos custos dos passos na Figura 3.2, e os valores de *hDLR* são apresentados na Figura 3.22. Observe, particularmente, que Bucareste aparece pela primeira vez na borda na etapa (e), mas não é selecionada para expansão porque seu f-custo (450) é maior que o de Pitesti (417). Outra maneira de dizer isso é que *pode* haver uma solução através de Pitesti, cujo custo seja tão baixo quanto 417, de modo que o algoritmo nao vai se contentar com uma solução que custe 450.

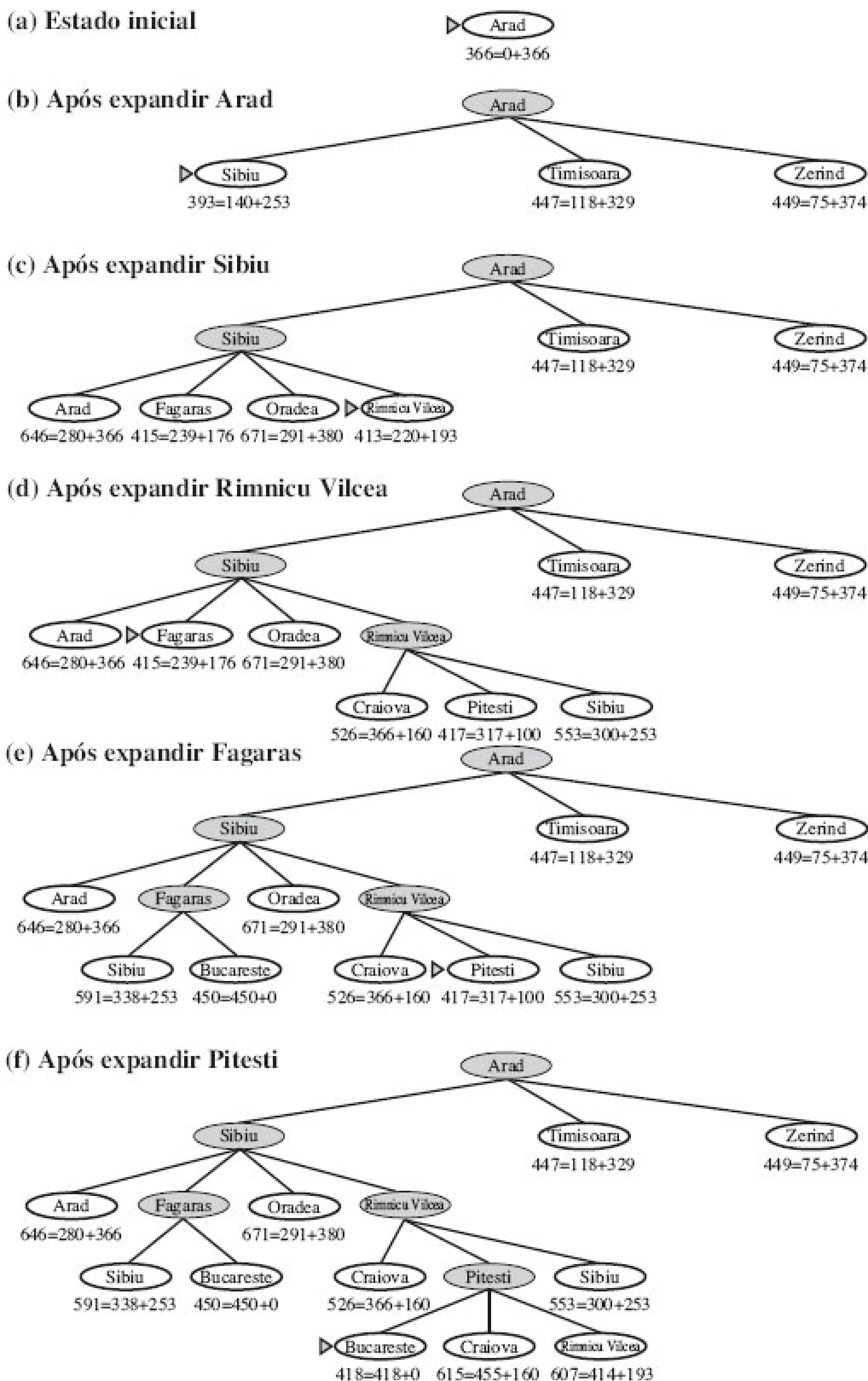


Figura 3.24 Etapas em uma busca A* para Bucareste. Nós são rotulados com $f = g + h$. Os valores de h são as distâncias em linha reta para Bucareste tomadas a partir da Figura 3.22.

Uma segunda condição um pouco mais forte chamada **consistência** (ou, algumas vezes, **monotonicidade**) é necessária apenas para aplicativos de A* para busca em grafos.[9] Uma heurística $h(n)$ será consistente se, para cada nó n e para todo sucessor n' de n gerado por uma ação a, o custo estimado de alcançar o objetivo de n não for maior do que o custo do passo de chegar a n' mais o custo estimado de alcançar o objetivo de n'.

$$h(n) \leq c(n, a, n') + h(n').$$

Essa é uma forma genérica de **desigualdade triangular**, que estipula que cada um dos lados de um triângulo não pode ser mais longo que a soma dos outros dois lados. Aqui, o triângulo é formado por *n, n′* e o objetivo *Gn* mais próximo de *n*. Para uma heurística admissível, a desigualdade faz todo sentido: se houvesse uma rota de *n* para *Gn* via *n′* que fosse mais barata do que *h*(*n*), que violasse a propriedade de que *h*(*n*) está em um limite inferior de custo para chegar a *Gn*.

É bastante fácil mostrar (Exercício 3.39) que toda a heurística consistente é também admissível. A consistência é, portanto, uma exigência mais rigorosa que a admissibilidade, mas deve-se trabalhar bastante para construir heurísticas que sejam admissíveis, mas não consistentes. Todas as heurísticas admissíveis que discutiremos neste capítulo são também consistentes. Considere por exemplo, *hDLR*. Sabemos que a desigualdade triangular genérica é satisfeita quando cada lado é medido pela distância em linha reta e que a distância em linha reta entre *n* e *n′* não é maior que *c(n, a, n′*). Assim, *hDLR* é uma heurística consistente.

Otimalidade de A*

Como mencionamos anteriormente, A* tem as seguintes propriedades: *a versão de busca em árvore de A* é ótima se h*(*n*) *for admissível, enquanto a versão de busca em grafos de A* é ótima se h*(*n*) *for consistente.*

Apresentaremos a segunda dessas duas afirmações, uma vez que é mais útil. O argumento se espelha essencialmente no argumento de otimalidade da busca de custo uniforme com *g* substituído por *f*, exatamente como definida no próprio algoritmo A*.

No primeiro passo iremos estabelecer o seguinte: *se h*(*n*) *for consistente, então os valores de f*(*n*) *ao longo de qualquer caminho serão não decrescentes*. A prova vem diretamente da definição de consistência. Suponha que *n′* seja sucessor de *n*, então $g(n') = g(n) + c(n, a, n')$ para alguma ação *a* e teremos:

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n).$$

O próximo passo será provar que sempre que A* *selecionar um nó n para expansão, o caminho ótimo para aquele nó foi encontrado*. Se isso não fosse verdade, deveria haver outro nó da borda *n′* no caminho ótimo do nó inicial até *n,* pela propriedade de separação de grafos da Figura 3.9 porque *f* é não decrescente ao longo de qualquer caminho, *n′* teria menor *f*-custo que *n* e teria sido selecionado em primeiro lugar.

Das duas observações anteriores, segue que a sequência de nós expandidos por A* utilizando a BUSCA_EM_GRAFOS está em ordem não decrescente de *f*(*n*). Assim, o primeiro nó objetivo selecionado para a expansão deve ser uma solução ótima porque *f* é o custo real para nós objetivo (que têm $h = 0$) e todos os outros nós objetivo visitados posteriormente terão custo igual ou maior.

O fato de que os custos *f* são não decrescentes ao longo de qualquer caminho significa que podemos também extrair **contornos** no espaço de estados, como os contornos de um mapa topográfico. A Figura 3.25 mostra um exemplo. Dentro do contorno rotulado 400, todos os nós têm *f*(*n*) menor ou igual a 400, e assim por diante. Então, devido a A* expandir o nó de borda de menor *f*-custo, podemos ver que uma busca A* espalha-se a partir do nó de início, adicionando nós em faixas

concêntricas de *f*-custo crescente.

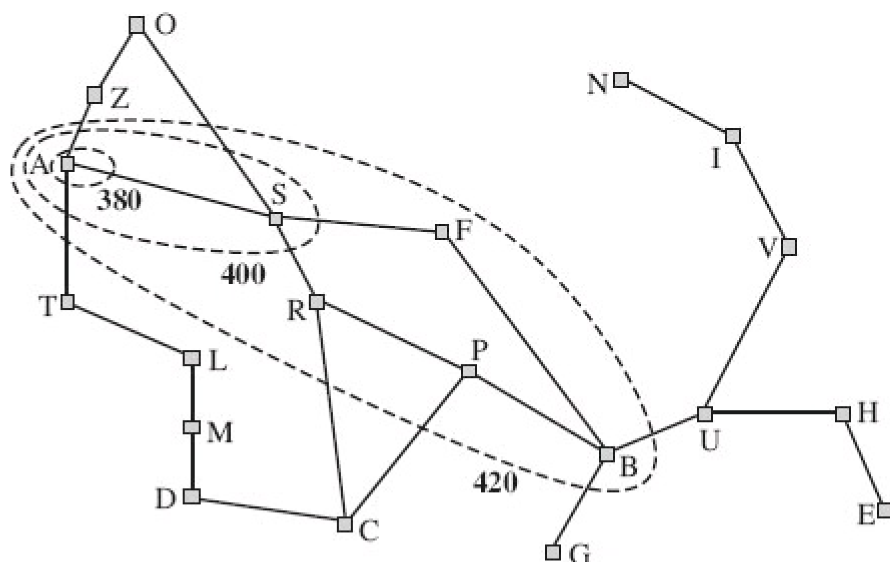


Figura 3.25 Mapa da Romênia mostrando contornos em $f = 380$, $f = 400$ e $f = 420$, com Arad como o estado inicial. Os nós dentro de um contorno dado tem *f*-custo menor ou igual ao valor do contorno.

Com busca de custo uniforme (busca A* utilizando $h(n) = 0$), as faixas serão "circulares" em torno do estado inicial. Com heurísticas mais precisas, as faixas irão se estender em direção ao estado objetivo e ficarão mais focadas em torno do caminho ótimo. Se C^* for o custo do caminho de solução ótima, então podemos dizer o seguinte:

- A* expande todos os nós com $f(n) < C^*$.
- A* pode então expandir alguns dos nós bem sobre o "contorno objetivo" *(onde* $f(n) = C^*$), antes de selecionar um nó objetivo.

A completeza requer que haja apenas um número finito de nós com custo menor ou igual a C^*, uma condição que é verdadeira se todos os custos de passo ultrapassarem alguns ϵ finitos e se b for finito.

Observe que A* não expande os nós com $f(n) > C^*$ — por exemplo, Timisoara não foi expandido na Figura 3.24, mesmo sendo um filho da raiz. Dizemos que a subárvore abaixo de Timisoara foi **podada**; porque *hDLR* é admissível, o algoritmo pode ignorar essa subárvore com segurança e ainda assim garantir otimalidade. O conceito de poda — eliminação de possibilidades de consideração sem ter de examiná-las — é importante para muitas áreas de IA.

Uma observação final é que, entre os algoritmos ótimos desse tipo — algoritmos que estendem os caminhos de busca a partir da raiz e utilizam a mesma informação heurística —, A* é **otimamente eficiente** para qualquer dado heurístico consistente. Ou seja, não é garantido que nenhum outro algoritmo ótimo expanda menos nós do que A* (exceto, possivelmente, através de desempate entre os nós com $f(n) = C^*$). Isso ocorre porque qualquer algoritmo que não expande todos os nós com $f(n) < C^*$ corre o risco de perder a solução ótima.

É muito bom que a busca A* seja completa, ótima e otimamente eficiente entre todos os algoritmos desse tipo. Infelizmente, isso não significa que A* seja a resposta para todas as nossas necessidades de busca. O problema é que, para a maioria dos problemas, o número de estados dentro do espaço de busca do contorno objetivo ainda é exponencial no comprimento da solução. Os detalhes da análise

estão fora do escopo deste livro, mas os resultados básicos são os seguintes. Para problemas com custos de passo constante, o crescimento em tempo de execução como uma função da solução ótima de profundidade d é analisada em termos do **erro absoluto** ou do **erro relativo** da heurística. O erro absoluto é definido como $\Delta \equiv h^* - h$, onde h^* é o custo real a partir da raiz para o objetivo, e o erro relativo é definido como $\in \equiv (h^* - h)/h^*$.

A complexidade resultante depende fortemente das suposições feitas sobre o espaço de estados. O modelo mais simples estudado é um espaço de estados que tem um único objetivo e é essencialmente uma árvore com ações reversíveis (um quebra-cabeças de oito peças satisfaz o primeiro e o terceiro desses pressupostos). Nesse caso, a complexidade de tempo de A* é exponencial no erro máximo absoluto, ou seja, $O(b^{\Delta})$. Para os custos de passo constante, podemos escrever isso como $O(b^{\in}d)$, onde d é a solução de profundidade. Para quase todas as heurísticas na prática, o erro absoluto é pelo menos proporcional ao custo do caminho h^*, por isso $\in$ é constante ou crescente e a complexidade de tempo é exponencial em d. Nós também podemos verificar o efeito de uma heurística mais precisa: $O(b^{\in}d) = O(b^{\in})d)$, então o fator de ramificação efetivo (mais formalmente definido na próxima seção) é $b^{\in}$.

Quando o espaço de estados tem muitos estados objetivos, particularmente estados objetivos *quase ótimos*, o processo de busca pode ser desviado do caminho ótimo e haverá um custo adicional proporcional extra ao número de objetivos cujo custo está dentro do fator $\in$ de custo ótimo. Finalmente, no caso geral de um grafo, a situação é ainda pior. Pode haver muitos estados exponencialmente com $f(n) < C^*$, mesmo que o erro absoluto esteja limitado por uma constante. Por exemplo, considere uma versão do mundo do aspirador de pó, onde o agente pode limpar qualquer quadrado pelo custo unitário, mesmo sem ter que visitá-lo: nesse caso, os quadrados podem ser limpos em qualquer ordem. Com N quadrados sujos inicialmente, existem 2^n estados em que um subconjunto foi limpo e todos eles estão no caminho da solução ótima — e, portanto, satisfazem $f(n) < C^*$, mesmo que a heurística tenha um erro de 1.

A complexidade de A* muitas vezes torna impraticável insistir em encontrar uma solução ótima. Variantes de A* que encontrem rapidamente soluções subótimas podem ser utilizadas ou, por vezes, ser projetadas heurísticas que sejam mais precisas, mas não necessariamente admissíveis. Em qualquer caso, o uso de uma boa heurística ainda oferece economia enorme em comparação com o uso de uma busca não informada. Na Seção 3.6 veremos a questão projetar uma boa heurística.

O tempo de computação, contudo, não é a principal desvantagem de A*. Por manter todos os nós gerados na memória (como fazem todos os algoritmos de BUSCA-EM-GRAFOS), a busca A* geralmente atinge um limite de memória ("estouro de memória") do espaço bem antes de exceder um dado limite de tempo. Por essa razão, A* não é praticável para muitos problemas de larga escala. Há, no entanto, algoritmos que superam o problema de espaço sem sacrificar a otimalidade ou integridade, a um custo pequeno em tempo de execução, conforme discutiremos a seguir.

3.5.3 Busca heurística limitada pela memória

A maneira mais simples para reduzir os requisitos de memória para A* é adaptar a ideia de aprofundamento iterativo para o contexto de busca heurística, resultando no algoritmo de

aprofundamento iterativo A* (IDA* – Iterative Deepening A*). A principal diferença entre IDA* e o aprofundamento iterativo padrão é que o corte utilizado é o *f*-custo (*g* + *h*) em vez da profundidade; em cada iteração, o valor de corte é o menor *f*-custo de qualquer nó que excedeu o corte na iteração anterior. O IDA* é prático para muitos problemas com custos de passo unitário e evita a sobrecarga associada à manutenção de uma fila ordenada de nós. Infelizmente, ele sofre das mesmas dificuldades dos custos de valor real que a versão iterativa da busca de custo uniforme descrita no Exercício 3.17. Esta seção examina brevemente outros dois algoritmos de memória limitada, chamados RBFS e MA*.

A **busca recursiva de melhor escolha** (RBFS – Recursive Best First Search) é um algoritmo recursivo simples que tenta imitar a operação de busca padrão pela melhor escolha, mas usando apenas um espaço linear de memória. O algoritmo é mostrado na Figura 3.26. Sua estrutura é semelhante ao de uma busca em profundidade recursiva, mas, em vez de continuar indefinidamente seguindo o caminho atual, ele utiliza a variável *f_limite* para acompanhar o *f*-valor do melhor caminho *alternativo* disponível de qualquer ancestral do nó atual. Se o nó atual exceder esse limite, a recursão reverte para o caminho alternativo. Com a reversão da recursão, o RBFS substitui o *f*-valor de cada nó ao longo do caminho por um **valor de backup** — o melhor *f*-valor de seus filhos. Dessa forma, a RBFS lembra o *f*-valor das melhores folhas da subárvore esquecida e pode, portanto, decidir se vale a pena reexpandir a subárvore algum tempo mais tarde. A Figura 3.27 mostra como a RBFS atinge Bucareste.

```
função BUSCA-RECURSIVA-PELA-MELHOR(problema) retorna uma solução ou falha
  retorna RBFS(problema, FAZ-NÓ(problema. ESTADO-INICIAL), ∞)
função RBFS (problema, nó, f_limite) retorna uma solução ou falha e um limite novo f_custo
  se problema. TESTE-OBJETIVO(nó.ESTADO) então retorne SOLUÇÃO (nó)
  sucessores ← [ ]
  para cada ação em problema. AÇÕES(nó.ESTADO) fazer
    adicionar NÓ-FILHO(problema, nó, ação) em sucessores
  se sucessores estiver vazio então retornar falha), ∞
  para cada s em sucessores fazer / * atualizar f com o valor da busca anterior, se houver */
    s.f ← max(s.g + s.h, nó.f))
  repita
    melhor ← valor f mais baixo do nó em sucessores
    se melhor.f > f_limite então retornar falha, melhor.f
    alternativa ← segundo valor f mais baixo entre sucessores
    resultado, melhor.f ← RBFS(problema, melhor, min(f_limite, alternativa)
    se resultado ≠ falha então retornar resultado
```

Figura 3.26 Algoritmo para busca de melhor escolha recursiva.

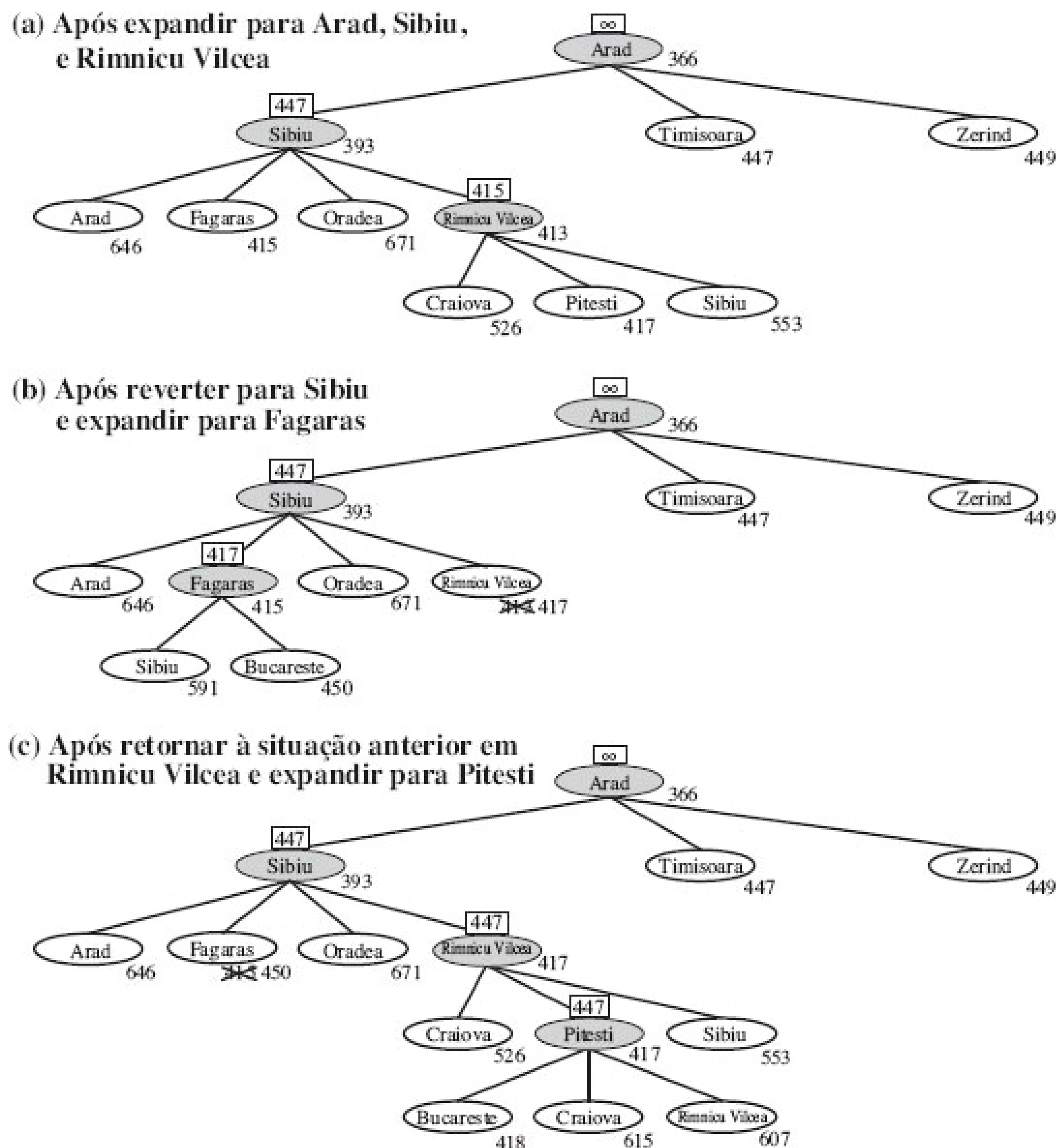


Figura 3.27 Etapas de uma busca RBFS para a rota mais curta para Bucareste. O valor do *f*_limite para cada chamada recursiva é mostrado no topo de cada nó atual, e cada nó é rotulado com seu *f*-custo. (a) O caminho via Rimnicu Vilcea é seguido até que a melhor folha atual (Pitesti) tenha um valor que é pior do que o melhor caminho alternativo (Fagaras). (b) A recursão reverte e o melhor valor da folha da subárvore esquecida (417) é copiado para Rimnicu Vilcea, então Fagaras é expandida, revelando um melhor valor da folha de 450. (c) A recursão reverte e o melhor valor da folha da subárvore esquecida (450) é copiado para Fagaras, então Rimnicu Vilcea é expandida. Dessa vez, devido ao melhor caminho alternativo (através de Timisoara), custa pelo menos 447, e a expansão continua para Bucareste.

A RBFS é um pouco mais eficiente do que a IDA*, mas ainda sofre pela geração excessiva de um mesmo nó. No exemplo da Figura 3.27, a RBFS segue o caminho via Rimnicu Vilcea, depois "muda de ideia" e tenta Fagaras, e depois muda de ideia novamente. Essas mudanças de ideia ocorrem porque, cada vez que o melhor caminho atual é estendido, seu *f*-valor possivelmente cresce — *h* geralmente é menos otimista para nós mais perto do objetivo. Quando isso acontece, o segundo melhor caminho pode se tornar o melhor caminho; assim, a busca tem que recuar para segui-lo. Cada mudança de ideia corresponde a uma iteração da IDA* e pode exigir muitas reexpansões de nós esquecidos para recriar o melhor caminho e estendê-lo com mais um nó.

Como a busca em árvore A*, a RBFS é um algoritmo ótimo se a função heurística *h*(*n*) for admissível. Sua complexidade de espaço é linear com relação à profundidade da solução ótima, mas a sua complexidade de tempo é bastante difícil de caracterizar: ela depende tanto da precisão da função heurística como do quão frequente o melhor caminho se altera à medida que os nós são expandidos.

As buscas IDA* e RBFS sofrem por usarem *pouca* memória. Entre iterações, a IDA* retém apenas um número único: o limite atual do *f*-custo. A RBFS retém mais informações na memória, mas utiliza apenas espaço linear: mesmo se mais memória estiver disponível, a RBFS não teria como fazer uso dela. Por esquecerem muito do que fizeram, ambos os algoritmos podem acabar reexpandindo os mesmos estados muitas vezes. Além disso, eles sofrem o crescimento potencialmente exponencial em complexidade associado com caminhos redundantes em grafos (veja a Seção 3.3).

Parece sensato, portanto, usar toda a memória disponível. Dois algoritmos que fazem isso são o **MA*** (A* de memória limitada) e o **SMA*** (MA* simplificado). O SMA* é bem mais simples, de modo que iremos descrevê-lo. O SMA* procede exatamente como o A*, expandindo a melhor folha até que a memória esteja cheia. Nesse ponto, não poderá adicionar um novo nó à árvore de busca sem suprimir um antigo. O SMA* sempre suprime o *pior* nó folha — o que tem o maior *f*_valor. Como o RBFS, o SMA*, em seguida, faz o backup do valor do nó esquecido em seu pai. Dessa forma, o ancestral de uma subárvore esquecida conhece a qualidade do melhor caminho daquela subárvore. Com essa informação, o SMA* regenera a subárvore somente quando todos os outros caminhos foram mostrados como piores do que o caminho que ele esqueceu. Outra maneira de dizer isso é que, se todos os descendentes de um nó *n* forem esquecidos, não saberemos para onde ir a partir de *n*, mas ainda teremos uma ideia de como vale a pena ir a algum lugar de *n*.

O algoritmo completo é muito complicado para reproduzir aqui,[10] mas há uma sutileza que vale a pena mencionar. Dissemos que o SMA* expande a melhor folha e exclui a pior folha. E, se *todos* os nós folha tiverem o mesmo *f*_valor? Para evitar a seleção do mesmo nó para exclusão e expansão, o SMA* expande a melhor folha mais *nova* e exclui a pior folha *mais antiga*. Estas coincidem quando há apenas uma folha, mas nesse caso a árvore de busca atual deve ser um único caminho da raiz até a folha que preenche toda a memória. Se a folha não for um nó objetivo, *mesmo que esteja em um caminho de solução ótima*, essa solução não será alcançável com a memória disponível. Desta forma, o nó poderá ser descartado exatamente como se não tivesse sucessores.

O SMA* estará completo se houver qualquer solução acessível, isto é, se *d*, a profundidade do nó objetivo mais raso, for menor que o tamanho da memória (expressa em nós). Será ótimo se qualquer solução ótima for alcançada; caso contrário, ele devolverá a melhor solução alcançável. Em termos práticos, o SMA* é uma escolha bastante robusta para encontrar soluções ótimas, especialmente quando o espaço de estados é um grafo, os custos de passo não são uniformes e a geração do nó é cara em comparação com a sobrecarga de manutenção da borda e do conjunto explorado.

Para problemas muito difíceis, no entanto, muitas vezes o SMA* é forçado a alternar constantemente entre muitos caminhos candidatos à solução, da qual pode caber na memória apenas um pequeno subconjunto (isso se assemelha ao problema de **degradação** em sistemas de paginação de disco). Então, o tempo extra que é necessário para a regeneração repetida dos mesmos nós significa que os problemas que poderiam ser praticamente solúveis com A*, dada a memória ilimitada, tornam-se intratáveis por SMA*. Isso significa dizer que *as limitações de memória podem*

se tornar um problema intratável do ponto de vista de tempo computacional. Embora nenhuma teoria atual explique o equilíbrio entre tempo e memória, esse parece ser um problema inevitável. A única saída é abandonar a exigência de otimalidade.

3.5.4 Aprendizagem para melhorar a busca

Apresentamos diversas estratégias fixas — busca em largura, busca gulosa de melhor escolha, e assim por diante — que foram projetadas por cientistas da computação. Um agente pode *aprender* para melhorar a busca? A resposta é sim, e o método se baseia em um conceito importante chamado de **espaço de estados do nível meta.** Cada estado em um espaço de estados do nível meta representa o estado (computacional) interno de um programa que faz a busca em um **espaço de estado do nível objeto**, tal como a Romênia. Por exemplo, o estado interno do algoritmo A* consiste na árvore de busca atual. Cada ação no espaço de estados do nível meta é um passo computacional que altera o estado interno; por exemplo, cada passo computacional em A* expande um nó folha e adiciona seus sucessores na árvore. Assim, a Figura 3.24, que mostra uma sequência de árvores de busca cada vez maiores, pode ser vista como representando um caminho no espaço de estados do nível meta onde cada estado no caminho é uma árvore de busca no nível objeto.

Agora, o caminho na Figura 3.24 tem cinco passos, incluindo um passo, a expansão de Fagaras, que não é especialmente útil. Para os problemas mais difíceis, haverá muitos erros desse tipo, e um algoritmo de **aprendizagem no nível meta** pode aprender com essas experiências para evitar explorar subárvores pouco promissoras. As técnicas utilizadas para esse tipo de aprendizagem são descritas no Capítulo 21. O objetivo da aprendizagem é minimizar o **custo total** da solução do problema, fazendo um compromisso entre o custo computacional e o custo do caminho.

3.6 FUNÇÕES HEURÍSTICAS

Nesta seção, examinaremos a heurística para o quebra-cabeças de oito peças, a fim de esclarecer a natureza da heurística em geral.

O quebra-cabeças de oito peças é um dos primeiros problemas de busca heurística. Como mencionado na Seção 3.2, o objetivo do quebra-cabeças é deslizar as peças horizontal ou verticalmente para o espaço vazio até que a configuração corresponda à configuração objetivo (Figura 3.28).

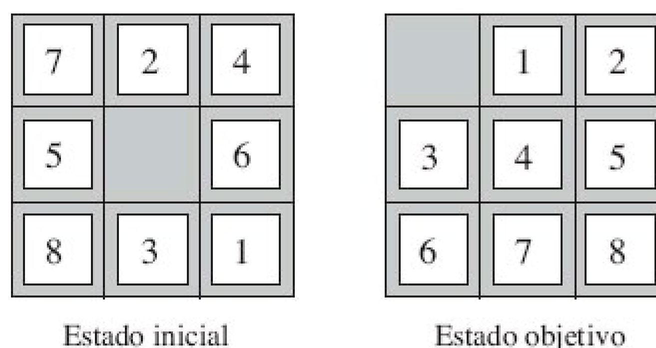


Figura 3.28 Exemplo típico de instância do quebra-cabeças de oito peças. A solução é de 26 passos longos.

O custo médio da solução para uma instância do quebra-cabeças de oito peças gerada aleatoriamente é de cerca de 22 passos. O fator de ramificação é cerca de 3 (quando a peça branca estiver no meio, é possível quatro movimentos, quando estiver em um canto, dois; quando estiver ao longo de uma borda, três). Isso significa que uma busca exaustiva da árvore de profundidade 22 ficaria em cerca de $3^{22} \approx 3{,}1 \times 10^{10}$ estados. Uma busca em grafos reduziria isso de um fator de cerca de 170.000, porque apenas 9!/2 = 181.440 estados distintos são alcançáveis (veja o Exercício 3.4). Esse é um número gerenciável, mas o número correspondente para o quebra-cabeças de 15 é de aproximadamente 10^{13}, assim a próxima tarefa é encontrar uma boa função heurística. Se quisermos encontrar as soluções mais curtas usando A*, precisamos de uma função heurística que nunca superestime o número de passos até o objetivo. Há um longo histórico de tais heurísticas para o quebra-cabeças de 15 peças. Seguem as duas mais utilizadas:

- h_1 = número de peças fora de lugar. Para a Figura 3.28, todas as oito peças estão fora de lugar, de modo que o estado inicial teria $h_1 = 8$. *h1* é uma heurística admissível porque é claro que qualquer peça que esteja fora de lugar deverá ser movida pelo menos uma vez.
- h_2 = soma das distâncias das peças de suas posições-objetivo. Devido às peças não poderem ser movidas ao longo de diagonais, a distância que vai contar é a soma das distâncias horizontal e vertical. Isso, às vezes, é chamado de **distância de quarteirão da cidade** ou **distância de Manhattan**. h_2 é também admissível porque todo movimento que pode ser feito move uma peça um passo mais perto do objetivo. As peças 1-8 no estado inicial dão uma distância de Manhattan de:

$$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18.$$

Como esperado, nenhuma delas superestima o custo da solução verdadeira, que é 26.

3.6.1 O efeito da precisão heurística sobre o desempenho

Uma forma de caracterizar a qualidade de uma heurística é o **fator de ramificação efetivo** b^*. Se o número total de nós gerados por A* para um problema particular for N e a solução da profundidade for d, então b^* é o fator de ramificação que uma árvore uniforme de profundidade d teria de ter a fim de conter $N + 1$ nós. Assim,

$$N+1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d.$$

Por exemplo, se A* encontra uma solução à profundidade 5 utilizando 52 nós, então o fator de ramificação efetivo é 1,92. O fator de ramificação efetivo pode variar entre instâncias do problema, mas geralmente é razoavelmente constante para problemas difíceis. (A existência de um fator de ramificação efetivo parte do resultado, mencionado anteriormente, implica que o número de nós expandidos por A* cresce exponencialmente com a solução da profundidade.) Portanto, as medidas

experimentais de b^* em um pequeno conjunto de problemas podem fornecer um bom guia para a utilidade geral da heurística. Uma heurística bem projetada teria um valor de b^* próximo de 1, permitindo que problemas bem grandes sejam resolvidos a um custo computacional razoável.

Para testar as funções heurísticas h_1 e h_2, geramos 1.200 problemas aleatórios com soluções que vão de 2 a 24 (100 para cada número par) e as resolvemos com busca de aprofundamento iterativo A* e com busca em árvore usando tanto h_1 como h_2. A Figura 3.29 apresenta o número médio de nós gerados por cada estratégia e o fator de ramificação efetivo. Os resultados sugerem que h_2 é melhor do que h_1, e é muito melhor do que utilizar busca de aprofundamento iterativo. Mesmo para problemas pequenos com $d = 12$, A* com h_2 é 50.000 vezes mais eficiente que a busca não informada de aprofundamento iterativo.

	Custo da Busca (nós gerados)			**Fator de Ramificação Efetivo**		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	-	539	113	-	1,44	1,23
16	-	1301	211	-	1,45	1,25
18	-	3056	363	-	1,46	1,26
20	-	7276	676	-	1,47	1,27
22	-	18094	1219	-	1,48	1,28
24	-	39135	1641	-	1,48	1,26

Figura 3.29 Comparação dos custos da busca e fatores de ramificação para a BUSCA-DE-APROFUNDAMENTO-ITERATIVO (IOS) e algoritimos de A* com h_1, h_2. Calcula-se a média dos dados sobre 100 exemplos do quebra-cabeças de oito peças para cada uma das diversas soluções que se distanciam de *d.*

A questão é se h_2 é sempre melhor que h_1. A resposta é: "Essencialmente, sim." É fácil verificar a partir das definições das duas heurísticas que, para qualquer nó n, $h_2(n) \geq h1(n)$. Dizemos portanto que h_2 **domina** h_1. A dominação se traduz diretamente em eficiência: A* utilizando h_2 nunca irá expandir mais nós do que A* utilizando h_1 (exceto, possivelmente, para alguns nós com $f(n) = C^*$). O argumento é simples. Lembre-se da observação de que cada nó com $f(n) < C^*$ certamente será expandido. É o mesmo que dizer que cada nó com $h(n) < C^* - g(n)$ certamente será expandido. Mas, devido a h_2 ser pelo menos tão grande quanto h_1 para todos os nós, cada nó que de fato for expandido

pela busca A* com h_2 será de fato também expandido com h_1, e h_1 poderá fazer com que outros nós também sejam expandidos. Por isso, geralmente é melhor utilizar uma função heurística com valores mais elevados desde que seja consistente e que o cálculo do tempo para a heurística não seja muito longo.

3.6.2 Geração de heurísticas admissíveis de problemas relaxados

Vimos que tanto h_1 (peças em lugares errados) como h_2 (distância de Manhattan) são heurísticas bastante boas para o quebra-cabeças de oito peças e que h_2 é melhor. Como h_2 apareceu? É possível para um computador inventar tal heurística mecanicamente?

h_1 e h_1 são estimativas do comprimento do caminho restante para o quebra-cabeças de oito peças, mas são também comprimentos de caminho perfeitamente precisos para versões simplificadas do quebra-cabeça. Se as regras do quebra-cabeça forem alteradas de modo que uma peça possa ser movida para qualquer lugar em vez de apenas para o quadrado adjacente vazio, então h_1 dará o número exato de passos para a solução mais curta. Da mesma forma, se uma peça puder ser movida um quadrado em qualquer direção, mesmo para um quadrado ocupado, h_2 dará o número exato de passos para a menor solução. Um problema com poucas restrições sobre as ações é chamado de **problema relaxado**. O grafo de espaço de estados do problema relaxado é um *supergrafo* do espaço de estados original porque a eliminação das restrições cria arestas adicionais no grafo.

Em razão do problema relaxado acrescentar arestas para o espaço de estados, qualquer solução ótima do problema original será, por definição, também uma solução do problema relaxado, mas o problema relaxado pode ter *melhores* soluções, se as arestas adicionadas fornecerem atalhos. Assim, *o custo de uma solução ótima para um problema relaxado é uma heurística admissível para o problema original.* Além disso, como a heurística derivada é o custo exato para o problema relaxado, este deverá obedecer à desigualdade triangular e, portanto, ser **consistente** (ver Seção 3.5.2).

Se a definição do problema for escrita em linguagem formal, é possível construir problemas relaxados automaticamente.[11] Por exemplo, se as ações do quebra-cabeças de oito peças forem descritas como

Uma peça pode se mover do quadrado A para B se
A for horizontal ou verticalmente adjacente a B **e** B estiver vazio,

podemos gerar três problemas relaxados, removendo uma ou ambas as condições:

(a) Uma peça pode se mover do quadrado A para o quadrado B se A for adjacente a B.
(b) Uma peça pode se mover do quadrado A para o quadrado B se B estiver vazio.
(c) Uma peça pode se mover do quadrado A para o quadrado B.

A partir de (a), podemos derivar h_2 (distância de Manhattan). O raciocínio é que h_2 seria a

pontuação adequada se movêssemos cada peça por vez para o seu destino. A heurística derivada de (b) será discutida no Exercício 3.31. A partir de (c), podemos derivar h_1 (peças em lugar errado) porque seria a pontuação adequada se as peças pudessem ser movidas para o destino final em um passo. Observe que é crucial que os problemas relaxados gerados por essa técnica possam ser resolvidos essencialmente *sem busca* porque as regras relaxadas permitem que o problema seja decomposto em oito subproblemas independentes. Se o problema relaxado for difícil de resolver, os valores da heurística correspondente serão caros para serem obtidos.[12]

Um programa chamado ABSOLVER pode gerar heurísticas automaticamente a partir de definições de problemas utilizando o método do "problema relaxado" e várias outras técnicas (Prieditis, 1993). O ABSOLVER gerou uma heurística nova para o quebra-cabeças de oito peças melhor do que qualquer heurística preexistente e encontrou a primeira heurística útil para o famoso quebra-cabeças do cubo de Rubik (também conhecido como cubo mágico).

Um problema com a geração de novas funções heurísticas é que muitas vezes não se consegue obter uma única heurística "claramente melhor". Se uma coleção de heurísticas admissíveis $h_1 \ldots hm$ estiver disponível para um problema e nenhuma delas dominar qualquer uma das outras, qual delas se deve escolher? Como se constata, não é preciso fazer uma escolha. Podemos ter o melhor dos mundos através da definição

$$h(n) = \max \{h_1(n), \ldots, \mathrm{h}m(n)\}.$$

Essa heurística composta utiliza qualquer função que seja mais precisa no nó em questão. Em razão das heurísticas da composição serem admissíveis, *h* é admissível, mas também é fácil provar que *h* é consistente. Além disso, *h* domina todos os seus componentes heurísticos.

3.6.3 Geração de heurísticas admissíveis de subproblemas: bancos de dados de padrões

As heurísticas admissíveis podem também ser derivadas da solução de custo de um **subproblema** do problema dado. Por exemplo, a Figura 3.30 mostra um subproblema da instância do quebra-cabeças de oito peças da Figura 3.28. O subproblema envolve levar as peças 1, 2, 3, 4 para suas posições corretas. O custo da solução ótima desse subproblema é claramente um limite inferior do custo do problema completo. Em alguns casos o custo de um subproblema pode ser mais preciso do que a distância de Manhattan.

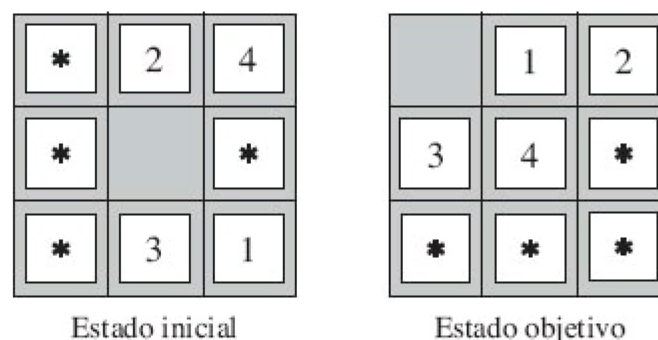


Figura 3.30 Um subproblema da instância do quebra-cabeças de oito peças apresentado na Figura 3.28. A tarefa é ter as peças 1, 2, 3 e 4 em suas posições corretas, sem se preocupar com o que acontece com as outras peças.

A ideia por trás **dos bancos de dados de padrões** é armazenar os custos exatos de solução para todas as instâncias possíveis do subproblema — em nosso exemplo, todas as configurações possíveis das quatro peças e da posição em branco (os locais das outras quatro peças são irrelevantes para fins de resolução do subproblema, mas a movimentação das peças será levada em conta em relação ao custo). Então vamos calcular uma heurística admissível *hBD* para cada estado completo encontrado durante uma busca, simplesmente pelo exame da configuração correspondente do subproblema no banco de dados. O próprio banco de dados é construído através de busca reversa[13] do objetivo e do registro do custo de cada novo padrão encontrado; o custo dessa busca é amortizado ao longo das muitas instâncias do problema.

A escolha de 1-2-3-4 é bastante arbitrária; também podemos construir bases de dados para 5-6-7-8, 2-4-6-8, e assim por diante. Cada banco de dados produz uma heurística admissível, e essas heurísticas podem ser combinadas, como explicado anteriormente, extraindo o valor máximo. Uma heurística combinada desse tipo é muito mais precisa do que a distância de Manhattan; o número de nós gerados na resolução aleatória do quebra-cabeças de 15 peças pode ser reduzido por um fator de 1.000.

A dúvida é se as heurísticas obtidas a partir do banco de dados 1-2-3-4 e 5-6-7-8 poderiam ser *somadas*, pois parece que os dois subproblemas não se sobrepõem. Será que isso ainda resultaria em uma heurística admissível? A resposta é não, porque as soluções dos subproblemas 1-2-3-4 e 5-6-7-8 para um dado estado quase certamente irão compartilhar alguns movimentos — é improvável que 1-2-3-4 possam ser movidos para seus lugares sem tocar em 5-6-7-8 e vice-versa. Mas, e se não contássemos esses movimentos? Ou seja, não registramos o custo total de resolver o subproblema 1-2-3-4, mas apenas o número de movimentos envolvendo 1-2-3-4. Então é fácil verificar que a soma dos dois custos ainda é um limite inferior sobre o custo de resolver todo o problema. Essa é a ideia por trás dos **bancos de dados de padrões disjuntos**. Com tais bancos de dados, é possível resolver aleatoriamente o quebra-cabeças de 15 peças em poucos milissegundos — o número de nós gerados é reduzido por um fator de 10.000 em comparação com a utilização da distância de Manhattan. Para um quebra-cabeças de 24 peças, pode ser obtido um aumento de eficiência de cerca de um fator de um milhão.

Os bancos de dados de padrões disjuntos funcionam para o quebra-cabeças de peças deslizantes porque o problema pode ser dividido de tal forma que cada movimento afete apenas um subproblema, pois apenas uma peça pode ser movimentada por vez. Para um problema como o do cubo de Rubik, esse tipo de subdivisão é difícil porque cada movimento afeta oito ou nove dos 26 cubos. Foram propostas formas mais gerais de definição de heurísticas aditivas admissíveis que se aplicam ao cubo de Rubik (Yang *et al.*, 2008), mas elas não resultaram em uma heurística melhor do que a melhor heurística não aditiva para o problema.

3.6.4 Aprendizagem de heurísticas a partir da experiência

Uma função heurística $h(n)$ deve ser capaz de estimar o custo de uma solução começando pelo estado do nó n. Como um agente poderia construir tal função? Nas seções anteriores foi dada uma solução, isto é, conceber problemas relaxados para os quais uma solução ótima pode ser facilmente encontrada. Outra solução é aprender com a experiência. "Experiência" aqui significa, por exemplo, resolver muitos quebra-cabeças de oito peças. Cada solução ótima para o problema do quebra-cabeças de oito peças fornece exemplos dos quais $h(n)$ pode ser aprendido. Cada exemplo consiste em um estado do caminho da solução e do custo real da solução a partir desse ponto. A partir desses exemplos, pode ser utilizado um algoritmo de aprendizagem para construir uma função $h(n)$ que pode (com sorte) prever os custos de solução para outros estados que surgirem durante a busca. As técnicas para fazer apenas isso utilizando redes neurais, árvores de decisão e outros métodos são demonstradas no Capítulo 18 (os métodos de aprendizagem por reforço descritos no Capítulo 21 também são aplicáveis).

Os métodos de aprendizagem indutiva funcionam melhor quando supridos de **características** de um estado que são relevantes para predizer o valor do estado, em vez de apenas uma simples descrição do estado. Por exemplo, a característica de "número de peças fora do lugar" pode ser útil em predizer a distância real de um estado a partir do objetivo. Vamos chamar essa característica de x_1 (n). Poderíamos extrair 100 configurações geradas aleatoriamente do quebra-cabeças de oito peças e reunir estatísticas sobre seus custos reais de solução. Podemos considerar que, quando $x_1(n)$ for 5, o custo médio de solução será cerca de 14, e assim por diante. Tendo em conta esses dados, o valor de x_1 poderá ser utilizado para prever $h(n)$. Certamente poderemos utilizar várias características. Uma segunda característica $x_2(n)$ pode ser o "número de pares de peças adjacentes que não são adjacentes no estado objetivo". Como deveríamos combinar $x_1(n)$ e $x_2(n)$ para prever $h(n)$? Uma abordagem comum é usar uma combinação linear:

$$h(n) = c_1x_1(n) + c_2x_2(n).$$

As constantes c_1 e c_2 são ajustadas para proporcionar o melhor ajuste para os dados reais sobre os custos da solução. Espera-se que tanto c_1 como c_2 sejam positivos porque as peças fora de lugar e os pares incorretos adjacentes tornam o problema mais difícil de resolver. Observe que essa heurística satisfaz a condição de $h(n) = 0$ para os estados-objetivo, mas não é necessariamente admissível ou consistente.

3.7 RESUMO

Este capítulo introduziu métodos que um agente pode usar para selecionar ações em ambientes determinísticos, observáveis, estáticos e completamente conhecidos. Em tais casos, o agente pode construir sequências de ações que alcançam seus objetivos; esse processo é chamado de **busca**.

- Antes de um agente poder começar a procurar soluções, ele deve identificar um **objetivo** e formular um **problema** bem definido.
- Um problema consiste em cinco partes: o **estado inicial**, um conjunto de **ações**, um **modelo de**

transição descrevendo os resultados dessas ações, uma função **teste de objetivo** e uma função **custo de caminho**. O ambiente do problema é representado por um **espaço de estados**. Um **caminho** pelo espaço de estados a partir do estado inicial até um estado objetivo é uma **solução**.

- Algoritmos de busca tratam estados e ações como atômicos: sem considerar qualquer estrutura interna que possam ter.
- Um algoritmo genérico de BUSCA-EM-ÁRVORE considera todos os caminhos possíveis para encontrar uma solução, enquanto um algoritmo de BUSCA-EM-GRAFO evita a consideração de caminhos redundantes.
- Os algoritmos de busca são analisados em termos de **completeza**, **otimização**, **complexidade de tempo** e **complexidade de espaço**. A complexidade depende de *b*, o fator de ramificação no espaço de estados, e de *d*, a profundidade da solução mais rasa.
- Métodos de busca não informados têm acesso apenas à definição do problema. Os algoritos básicos são os seguintes:
  - A **busca em largura** seleciona para expansão os nós mais rasos; ela é completa, ótima para passos de custo unitário, mas tem complexidade de tempo exponencial.
  - A **busca de custo uniforme** expande o nó com o menor custo de caminho, $g(n)$ e é ótima para passos de custos genéricos.
  - A **busca em profundidade** expande o nó não expandido mais profundo. Ela não é completa nem ótima, mas tem complexidade espacial linear. A **busca em profundidade limitada** adiciona um limite em profundidade.
  - A **busca de aprofundamento iterativo** chama a busca em profundidade com limites crescentes de profundidade até encontrar um objetivo. Ela é completa, ótima para passos de custo unitário, tem complexidade de tempo comparável à busca em largura e tem complexidade de espaço linear.
  - A **busca bidirecional** pode reduzir enormemente a complexidade de tempo, mas nem sempre é aplicável e pode exigir muito espaço.
- Os métodos de busca informada podem ter acesso a uma função **heurística** $h(n)$ que estima o custo de uma solução a partir de n.
  - O algoritmo geral de **busca de melhor** escolha seleciona um nó para a expansão de acordo com uma **função de avaliação.**
  - A **busca gulosa** de melhor escolha expande os nós com $h(n)$ mínimo. Não é ótima, mas pode ser eficiente.
  - Uma **busca A*** expande os nós com $f(n) = g(n) + h(n)$ mínimo. A* é completa e ótima, desde que $h(n)$ seja admissível (para a BUSCA-EM-ÁRVODE) ou consistente (para a BUSCA-EM-GRAFO). A complexidade de espaço do A* ainda é proibitiva.
  - **RBFS** (busca recursiva de melhor escolha) e **SMA*** (A* de memória limitada simplificada) são algoritmos robustos, de busca ótima, que utilizam porções limitadas de memória; dado um tempo suficiente, podem resolver os problemas que A* não pode resolver, por ficar sem memória.
- O desempenho de algoritmos de busca heurística depende da qualidade da função heurística. Pode-se, por vezes, construir uma boa heurística, através do relaxamento da definição do problema, armazenando os custos de solução pré-computados dos subproblemas em um banco de dados de padrões ou aprendendo a partir da experiência com uma classe de problemas.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O tema busca em espaço de estados teve origem mais ou menos em sua forma atual nos primórdios da IA. O trabalho de Newell e Simon sobre a Logic Theorist (1957) e o GPS (1961) levou ao estabelecimento dos algoritmos de busca como as armas fundamentais do arsenal dos pesquisadores de IA da década de 1960 e ao estabelecimento da resolução de problemas como tarefa canônica da IA. O trabalho em pesquisa operacional por Richard Bellman (1957) mostrou a importância dos custos de caminho aditivo na simplificação dos algoritmos de otimização. O texto *Automated Problem Solving* de Nils Nilsson (1971) estabeleceu a área sobre uma base teórica sólida.

A maior parte dos problemas de busca em espaço de estados analisados neste capítulo tem uma longa história na literatura e são menos triviais do que parecem ser. O problema dos missionários e canibais usado no Exercício 3.9 foi analisado em detalhes por Amarel (1968). Ele foi considerado anteriormente em IA por Simon e Newell (1961) e em pesquisa operacional por Bellman e Dreyfus (1962).

O quebra-cabeça de oito peças é um irmão menor do quebra-cabeça de 15 peças, cuja história foi contada extensamente por Slocum e Sonneveld (2006). Acreditava-se amplamente ter sido inventada pelo famoso projetista de jogos americano Sam Loyd, com base em suas alegações sobre os resultados a partir de 1891 (Loyd, 1959). Na verdade, foi inventado por Noyes Chapman, um agente de correio em Canastota, Nova York, em meados da década de 1870. (Chapman foi incapaz de patentear sua invenção, a patente genérica foi concedida a Ernest Kinsey em 1878, abrangendo blocos deslizantes com letras, números ou imagens.) Rapidamente atraiu a atenção do público e dos matemáticos (Johnson e Story, 1879; Tait, 1880). Os editores do *American Journal of Mathematics* declararam: "Nas últimas semanas, o quebra-cabeça de 15 peças chegou ao público americano e pode-se dizer com segurança que ele atraiu a atenção de nove entre dez pessoas de ambos os sexos e de todas as idades e condições sociais na comunidade." Ratner e Warmuth (1986) mostraram que a versão geral de $n \times n$ do quebra-cabeça pertence à classe de problemas NP-completos.

O problema de oito rainhas foi originalmente publicado anonimamente na revista alemã de xadrez *Schach* em 1848; mais tarde, ele foi atribuído a um certo Max Bezzel. O problema foi republicado em1850 e, nessa época, atraiu a atenção do eminente matemático Carl Friedrich Gauss, que tentou enumerar todas as soluções possíveis. Inicialmente ele achou apenas 72, mas eventualmente achou a resposta correta de 92, apesar de Nauck ter publicado primeiro todas as 92 soluções, em 1850. Netto (1901) generalizou o problema para n rainhas, e Abramson e Yung (1989) encontraram um algoritmo $O(n)$.

Cada um dos problemas de busca do mundo real listado nesse capítulo foi assunto de um grande esforço de pesquisa. Os métodos para selecionar voos ótimos de linhas aéreas permanecem patenteados em sua maior parte, mas Carl de Marcken (em comunicação pessoal) mostrou que a cotação e as restrições de passagens de linhas aéreas se tornaram tão complicadas que o problema de selecionar um voo ótimo é formalmente *indecidível*. O problema do caixeiro-viajante (TSP) é um problema combinatório-padrão em ciência da computação teórica (Lawler *et al*., 1992). Karp (1972) provou que o TSP é NP-difícil, mas foram desenvolvidos métodos efetivos de aproximação heurística (Lin e Kernighan, 1973). Arora (1998) criou um esquema de aproximação completamente polinomial para TSPs euclidianos. Os métodos de leiaute de VLSI foram pesquisados por Shahookar

e Mazumder (1991), e surgiram muitos artigos em periódicos de otimização de leiaute de VLSI. Os problemas de navegação e montagem de robôs são discutidos no Capítulo 25.

Os algoritmos de busca sem informação para resolução de problemas constituem um tópico central da ciência de computação clássica (Horowitz e Sahni, 1978) e da pesquisa operacional (Dreyfus, 1969). A busca em largura foi formulada para resolver labirintos por Moore (1959). O método de **programação dinâmica** (Bellman, 1957), que registra sistematicamente soluções para todos os subproblemas de comprimentos crescentes, pode ser visto como uma forma de busca em largura sobre grafos. O algoritmo de caminhos mais curtos de dois pontos de Dijkstra (1959) é a origem da busca de custo uniforme. Esses trabalhos também apresentaram a ideia de conjuntos explorados e de borda (listas abertas e fechadas).

Uma versão de aprofundamento iterativo projetada para faser uso eficiente do relógio no xadrez foi usada primeiro por Slate e Atkin (1977) no programa de jogo de xadrez CHESS 4.5. O algoritmo B de Martelli (1977) inclui um aspecto de aprofundamento iterativo e também domina o pior caso de desempenho de heurística admissível, mas inconsistente, de A*. A técnica do aprofundamento iterativo veio à tona no trabalho de Korf (1985a). A busca bidirecional, que foi apresentada por Pohl (1971), também pode ser eficaz em alguns casos.

O uso da informação heurística na resolução de problemas apareceu em um ensaio inicial por Simon e Newell (1958), mas a frase "busca heurística" e o uso de funções heurísticas que estimam a distância até o objetivo vieram um pouco mais tarde (Newell e Ernst, 1965; Lin, 1965). Doran e Michie (1966) realizaram estudos experimentais extensos de busca heurística. Embora tenham analisado o comprimento do caminho e a "penetrância" (a razão entre o comprimento do caminho e o número total de nós examinados até então), eles parecem ter ignorado as informações fornecidas pelo caminho de menor custo $g(n)$. O algoritmo A*, incorporando o custo do caminho atual em busca heurística, foi desenvolvido por Hart, Nilsson e Raphael (1968), com algumas correções posteriores (Hart *et al*., 1972). Dechter e Pearl (1985) demonstraram a eficiência ótima de A*.

O ensaio original de A* introduziu a condição de consistência sobre as funções heurísticas. A condição monotone foi introduzida por Pohl (1977) como uma simples substituição, mas Pearl (1984) mostrou que os dois eram equivalentes.

Pohl (1977) foi pioneiro no estudo da relação entre o erro em funções heurísticas e a complexidade do tempo de A*. Os resultados básicos foram obtidos para a de busca em árvore com custo do passo unitário e um nó objetivo único (Pohl, 1977; Gaschnig, 1979; Huyn *et al*., 1980; Pearl, 1984) e com múltiplos nós objetivos (Dinh *et al*., 2007). O "fator de ramificação efetivo" foi proposto por Nilsson (1971) como medida empírica da eficiência que é equivalente a assumir o custo de tempo de $O((b^*)^d)$. Para a busca em árvore aplicada a um grafo, Korf *et al*. (2001) argumentam que o custo do tempo é mais bem modelado como $O(b^{d-k})$, onde k depende da precisão heurística; no entanto, essa análise tem suscitado alguma controvérsia. Para a busca em grafos, Helmert e Röger (2008) observaram que vários problemas conhecidos continham um número exponencial de nós em caminhos da solução ótima, implicando complexidade de tempo exponencial para A*, mesmo com erro absoluto constante em h.

Existem muitas variações sobre o algoritmo A*. Pohl (1973) propôs o uso de *ponderação dinâmica*, que usa uma soma ponderada $fw(n) = wgg(n) + whh(n)$ do comprimento do caminho atual e da função heurística como uma função de avaliação, em vez da simples soma $f(n) = g(n) + h(n)$

utilizada em A*. Os pesos *wg* e *wh* são ajustados dinamicamente conforme a busca avança. O algoritmo de Pohl pode ser demonstrado como $\in$-admissível, ou seja, garante encontrar soluções com um fator 1 + $\in$ da solução ótima, onde $\in$ é um parâmetro fornecido ao algoritmo. A mesma propriedade é exibida pelo algotitmo $A^*_\in$ (Pearl, 1984), que pode selecionar qualquer nó da borda desde que a sua relação *f*-custo esteja dentro de um fator 1 + $\in$ do nó da borda de menor *f*_custo. A seleção pode ser feita de modo a minimizar o custo da busca.

Versões bidirecionais de A* têm sido investigadas; uma combinação de A* bidirecional e pontos de referência conhecidos foi utilizada para encontrar rotas de forma eficiente para o serviço de mapas on-line da Microsoft (Goldberg *et al*., 2006). Após coletar um conjunto de caminhos entre os pontos de referência, o algoritmo pode encontrar um caminho ótimo entre qualquer par de pontos em um grafo de 24 milhões de pontos dos Estados Unidos, buscando em menos de 0,1% do grafo. Outras abordagens para a busca bidirecional incluem a busca em largura que retrocede a partir do primeiro objetivo até uma profundidade fixa, seguido por uma busca IDA* para a frente (Dillenburg e Nelson, 1994; Manzini, 1995).

O algoritmo A* e outras buscas de espaço de estados estão intimamente relacionados com as técnicas *branch-and-bound* que são amplamente utilizadas em pesquisa operacional (Lawler e Wood, 1966). As relações entre busca de espaço de estados e branch-and-bound têm sido investigadas em profundidade (Kumar e Kanal, 1983; Nau *et al*., 1984; Kumar *et al*., 1988). Martelli e Montanari (1978) demonstram uma ligação entre programação dinâmica (ver o Capítulo 17) e certos tipos de busca de espaço de estados. Kumar e Kanal (1988) tentaram uma "grande unificação" da busca heurística, programação dinâmica e técnicas de branch-and-bound sob o nome de PDC — "processo de decisão composto".

Como os computadores no final dos anos 1950 e início dos anos 1960 tinham no máximo alguns milhares de palavras de memória principal, a memória limitada da busca heurística foi um tema de pesquisa inicial. O Graph Traverser (Doran e Michie, 1966), um dos primeiros programas de busca, passa para um operador após buscar pela melhor escolha até o limite de memória. O IDA* (Korf, 1985a, 1985b) foi o primeiro algoritmo ótimo de busca heurística com memória limitada usado amplamente e desde então tem sido desenvolvido um grande número de variantes. Uma análise da eficiência do IDA* e de suas dificuldades com heurística de valor real aparece em Patrick *et al*. (1992).

O RBFS (Korf, 1993) é realmente um pouco mais complicado do que o algoritmo mostrado na Figura 3.26, que é mais próximo de um algoritmo desenvolvido de forma independente chamado de **expansão iterativa** (Russell, 1992). O RBFS utiliza o limite inferior, bem como o superior; os dois algoritmos comportam-se de forma idêntica com heurísticas admissíveis, mas o RBFS expande os nós em ordem da melhor escolha, mesmo com uma heurística não admissível. A ideia de acompanhar o melhor caminho alternativo apareceu anteriormente na implementação elegante de Prolog do A* de Bratko (1986) e no algoritmo DTA* (Russell e Wefald, 1991). O último trabalho também discute os espaços de estados em metanível e aprendizagem em metanível.

O algoritmo MA* apareceu em Chakrabarti *et al.* (1989). O SMA*, ou MA* simplificado, surgiu de uma tentativa de implementar MA* como um algoritmo de comparação para o IE (Russell, 1992). Kaindl e Khorsand (1994) aplicaram SMA* para produzir um algoritmo de busca bidirecional que é substancialmente mais rápido que os algoritmos anteriores. Korf e Zhang (2000) descreveram uma

abordagem de divisão e conquista, e Zhou e Hansen (2002) introduziram a busca em grafos de A* de memória limitada e uma estratégia para mudar para busca em largura e aumentar a eficiência da memória (Zhou e Hansen, 2006). Korf (1995) fez uma resenha as técnicas de busca de memória limitada.

A ideia de que heurísticas admissíveis podem ser derivadas de relaxamento de problema surgiu no ensaio seminal de Held e Karp (1970), que utilizou a heurística da árvore geradora mínima para resolver o TSP (ver o Exercício 3.30).

A automação do processo de relaxamento foi implementada com sucesso por Prieditis (1993), com base no trabalho anterior com Mostow (Mostow e Prieditis, 1989). Holte e Hernadvolgyi (2001) descreveram as etapas mais recentes para automatizar o processo. O uso de bases de dados de padrões para derivar heurísticas admissíveis é devido a Gasser (1995) e Culberson e Schaeffer (1996, 1998); bancos de dados de padrões disjuntos são descritos por Korf e Felner (2002); um método similar, usando padrões simbólicos, é devido a Edelkamp (2009). Felner *et al.* (2007) mostraram como compactar bancos de dados de padrões para economizar espaço. A interpretação probabilística da heurística foi investigada em profundidade por Pearl (1984) e Hansson e Mayer (1989).

De longe a fonte mais abrangente sobre algoritmos de busca heurística e heurística é o texto *Heuristics* de Pearl (1984). Esse livro oferece uma cobertura especialmente boa da grande variedade de ramificações e variações de A*, incluindo provas rigorosas de suas propriedades formais. Kanal e Kumar (1988) apresentaram uma antologia de artigos importantes sobre busca heurística, e Rayward-Smith *et al*. (1996) cobriram abordagens da pesquisa operacional. Artigos sobre novos algoritmos de busca, que curiosamente continuam a ser descobertos, aparecem em revistas como *Artificial Inteligence* e *Journal of the ACM*.

O tema algoritmos de **busca paralela** não foi abordado no capítulo, em parte, porque exige uma longa discussão sobre arquiteturas de computadores paralelos. A busca paralela tornou-se um tema popular na década de 1990, tanto em IA como em teoria da ciência da computação (Mahanti e Daniels, 1993; Grama e Kumar, 1995; Crauser *et al*., 1998) e está voltando na era das novas arquiteturas multicore e cluster (Ralphs *et al*., 2004; Korf e Schultze, 2005). Os algoritmos de busca para grafos muito grandes que requerem armazenamento em disco (Korf, 2008) também apresentam importância crescente.

EXERCÍCIOS

3.1 Explique por que a formulação do problema deve seguir a formulação do objetivo.

3.2 O objetivo é dirigir o robô para fora de um labirinto. O robô inicia no meio do labirinto em direção ao norte. Você pode virar o robô em direção ao norte, sul, leste ou oeste. O robô pode ser comandado para mover uma certa distância para frente, apesar que irá parar antes de bater no muro.

a. Formule esse problema. Qual é o tamanho do espaço de estados?

b. Ao navegar pelo labirinto, é necessário virar apenas na interseção de dois ou mais corredores. Reformule esse problema usando essa observação. Qual será o tamanho do espaço de estados

agora?

c. De qualquer ponto do labirinto, podemos mover em qualquer uma das quatro direções, até ter alcançado um ponto de virar e essa é a única ação que precisa ser feita. Reformule o problema usando essas ações. É necessário acompanhar a orientação do robô para resolver esse problema?

d. Na descrição inicial do problema já abstraímos do mundo real, restringindo as ações e removendo os detalhes. Liste três simplificações que fizemos.

3.3 Suponha que dois amigos vivam em cidades de locais diferentes em um mapa, tais como mostra o mapa da Romênia na Figura 3.2. Podemos mover cada amigo de cada vez simultaneamente para uma cidade vizinha no mapa. A quantidade de tempo necessário para se deslocar da cidade i à vizinha j é igual à distância da estrada $d(i,j)$ entre as cidades, mas a cada vez que o amigo chega primeiro deve esperar até o outro chegar (e telefonar para o primeiro do celular) antes que comece a próxima vez de se movimentarem. Queremos que os dois amigos se encontrem o mais rápido possível.

a. Escreva uma formulação detalhada para esse problema de busca (será útil definir alguma notação formal).

b. Seja $D(i, j)$ a distância em linha reta entre as cidades i e j. Qual das seguintes funções heurísticas é admissível? (i) $D(i, j)$, (ii) $2 \cdot D(i, j)$, (iii) $D(i, j)/2$.

c. Há mapas completamente conectados para os quais não existe solução?

d. Há mapas em que todas as soluções requerem que um amigo visite a mesma cidade duas vezes?

3.4 Mostre que os estados do quebra-cabeça de oito peças se dividem em dois conjuntos disjuntos, tais que qualquer estado seja acessível a partir de qualquer outro estado no mesmo conjunto, enquanto nenhum estado pode ser acessado de qualquer outro estado no outro conjunto. (*Dica:* Veja Berlekamp *et al.* (1982).) Elabore um procedimento para decidir em que conjunto um dado estado se encontra e explique por que isso é útil para gerar estados aleatórios.

3.5 Considere o problema de n rainhas usando a formulação incremental "eficiente" dada na página 72. Explique por que o tamanho do espaço de estados é pelo menos $\sqrt[3]{n!}$ e faça uma estimativa do maior n para o qual a exploração exaustiva é possível. (*Dica:* Derive um limite inferior sobre o fator de ramificação, considerando o número máximo de quadrados que uma rainha pode atacar em qualquer coluna.)

3.6 Forneça uma formulação completa do problema para cada um dos seguintes itens. Escolha a formulação suficientemente precisa para ser implementada.

a. Usando apenas 4 cores, colorir um mapa plano de tal forma que duas regiões adjacentes não tenham a mesma cor.

b. Um macaco com 30 cm está em uma sala onde tem algumas bananas suspensas em um teto de 80 cm. Ele gostaria de pegar as bananas. A sala contém dois engradados móveis e escaláveis com 30 cm de altura que podem ser empilhados.

c. Existe um programa que exibe a mensagem "registro de entrada inválido" ao alimentar determinado arquivo com registros de entrada. Você sabe que o processamento de cada registro é independente de outros registros e deseja descobrir qual registro é inválido.

d. Existem três jarras que medem 12, 8 e 3 galões e uma torneira de água. As jarras podem ser enchidas ou esvaziadas uma da outra ou para o chão. Medir exatamente um galão, somente com essas operações.

3.7 Considere o problema de encontrar o caminho mais curto entre dois pontos em um plano que tem obstáculos poligonais convexos, como mostra a Figura 3.31. Essa é uma idealização do problema que um robô tem de resolver para navegar em um ambiente congestionado.

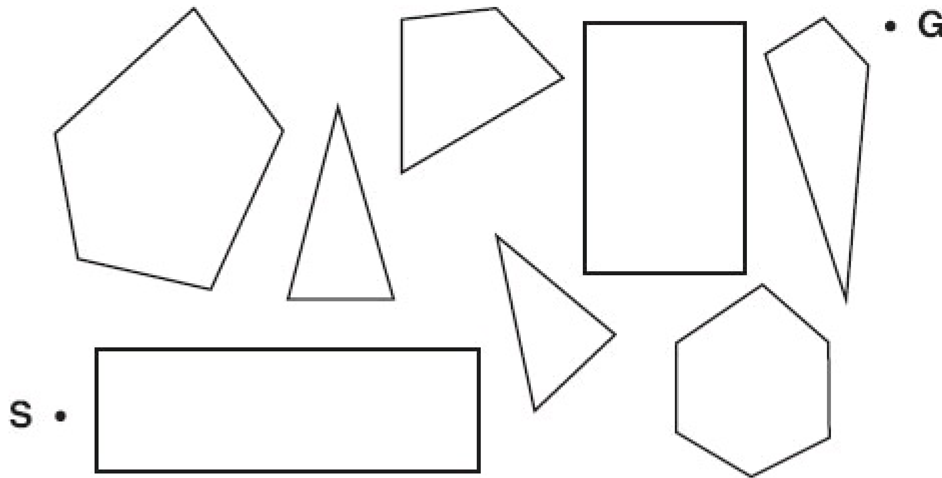


Figura 3.31 Um cenário com objetos poligonais. S e G são os estados de partida e objetivo.

a. Suponha que o espaço de estados consista de todas as posições (x, y) do plano. Quantos estados existem? Quantos caminhos existem até o objetivo?

b. Explique brevemente por que o caminho mais curto de um vértice de um polígono até qualquer outro vértice na cena deve consistir de segmentos de reta que unem alguns vértices dos polígonos. Agora defina um bom espaço de estados. Qual é o tamanho desse espaço de estados?

c. Defina as funções necessárias para implementar o problema de busca, incluindo uma função AÇÕES que receba um vértice como entrada e devolve um conjunto de vetores, sendo que cada vetor mapeia o vértice atual a um dos vértices que pode ser alcançado por uma linha reta. (Não se esqueça dos vizinhos no mesmo polígono.) Utilize a distância em linha reta como função heurística.

d. Aplique um ou mais algoritmos deste capítulo para resolver alguns problemas nesse domínio e comente seu desempenho.

3.8 Na Seção 3.1.1, dissemos anteriormente que não consideraríamos os problemas com custos de caminhos negativos. Neste exercício, vamos explorar esse tema com maior profundidade.

a. Suponha que as ações possam ter custos negativos arbitrariamente grandes; explique por que essa possibilidade forçaria qualquer algoritmo ótimo a explorar todo o espaço de estados.

b. Ajudaria se insistíssemos no fato de que os custos dos passos devem ser maiores ou iguais a alguma constante negativa c? Considere tanto a busca em árvores como em grafos.

c. Suponha que um conjunto de ações forme um laço no espaço de estados de modo que a cada execução dessas ações em alguma ordem não resulte em mudança no estado. Se todas essas ações tiverem custo negativo, qual será a implicação desse fato sobre o comportamento ótimo de um agente em tal ambiente?

d. É possível imaginar facilmente ações com custo negativo alto, até mesmo em domínios como o de roteamento. Por exemplo, alguns trechos da estrada poderiam ter belas paisagens que superem de longe os custos normais em termos de tempo e combustível. Explique, em termos precisos, dentro do contexto de busca em espaços de estados, por que os seres humanos não dirigem indefinidamente em ciclos por belos cenários e explique como definir o espaço de estados e ações para roteamento, de forma que agentes artificiais também possam evitar ciclos repetitivos.

e. Você pode imaginar um domínio real em que os custos dos passos sejam de tal forma que provoquem a entrada em ciclos repetitivos?

3.9 O problema de **missionários e canibais** é normalmente enunciado como a seguir. Três missionários e três canibais estão em um lado de um rio, juntamente com um barco que pode levar uma ou duas pessoas. Descubra um meio de fazer todos atravessarem o rio sem deixar que um grupo de missionários de um lado fique em número menor que o número de canibais nesse mesmo lado do rio. Esse problema é famoso em IA porque foi assunto do primeiro artigo que abordou a formulação de problemas a partir de um ponto de vista analítico (Amarel, 1968).

a. Formule o problema precisamente, fazendo apenas as especificações necessárias para assegurar uma solução válida. Faça um diagrama do espaço de estados completo.

b. Implemente e resolva o problema de forma ótima, utilizando um algoritmo de busca apropriado. É uma boa ideia verificar a existência de estados repetidos?

c. Por que você imagina que as pessoas têm dificuldades para resolver esse quebra-cabeça, considerando que o espaço de estados é tão simples?

3.10 Defina com suas próprias palavras os seguintes termos: estado, espaço de estados, árvore de busca, nó de busca, objetivo, ação, modelo de transição e fator de ramificação.

3.11 Qual é a diferença entre um estado do mundo, uma descrição do estado e um nó de busca? Por que é útil essa distinção?

3.12 Uma ação tal como *Ir(Sibiu)* consiste realmente em uma longa sequência de ações mais refinadas: ligar o carro, soltar o freio, acelerar para a frente etc. Ter ações compostas desse tipo reduz o número de passos em uma sequência de soluções, reduzindo assim o tempo de busca. Suponha que tomemos essa lógica ao extremo, construindo ações supercompostas de todas as sequências possíveis de ações *Ir*. Assim, cada instância do problema é resolvida por uma única ação supercomposta, como *Ir(Sibiu)Ir(Rimnicu Vilcea)Ir(Pitesti)Ir(Bucareste)*. Explique como a busca trabalharia nessa formulação. Essa é uma abordagem prática para acelerar a resolução do problemas?

3.13 Prove que a BUSCA EM GRAFO satisfaz a propriedade de separação do grafo ilustrada na Figura 3.9 (Dica: comece mostrando que a propriedade se mantém no início. Depois mostre que se ela se mantém antes da iteração com o algoritmo, mantém-se também depois.) Descreva um algoritmo de busca que viole a propriedade.

3.14 Qual das seguintes alternativas são falsas e quais são verdadeiras? Explique suas respostas.

a. A busca em profundidade sempre expande pelo menos tantos nós quanto a busca A* com uma

heurística admissível.

b. h(n) = 0 é uma heurística admissível para o quebra cabeças de 8 peças.

c. Em robótica, A* não é útil porque as percepções, estados e ações são contínuas.

d. A busca em largura é completa mesmo se os custos de passos iguais a zero forem permitidos.

e. Assuma que a torre pode se mover em um tabuleiro de xadrez qualquer quantidade de quadrados em linha reta, verticalmente ou horizontalmente, mas não pode pular sobre as peças. A distância de Manhattan é uma heurística admissível para o problema de movimentar a torre do quadrado A para o B no menor número de movimentos.

3.15 Considere um espaço de estados onde o estado inicial é o número 1 e cada estado k tem dois sucessores: números 2k e 2k+1.

a. Represente a porção do espaço de estados para os estados de 1 a 15.

b. Suponha que o estado objetivo seja 11. Liste a ordem em que os nós serão visitados pela busca em largura, busca em profundidade limitada com o limite 3 e busca de aprofundamento iterativo.

c. Como a busca bidirecional funcionaria nesse problema? Qual é o fator de ramificação em cada direção?

d. A resposta ao item (c) sugere uma reformulação do problema que permitiria resolver o problema de ir do estado 1 para um determinado estado objetivo com quase nenhuma busca?

e. Chame a ação que vai de k para 2k de Esquerda e a ação de k que vai para 2k + 1 de Direita.É possível encontrar um algoritmo que devolva a solução desse problema absolutamente sem nenhuma busca?

3.16 Uma ferrovia de brinquedo, com trilhos de madeira, contém as peças mostradas na Figura 3.32. A tarefa é conectar essas peças em uma estrada de ferro, sem trilhos sobrepostos e pontas soltas, onde um trem poderia descarrilhar, saindo dos trilhos.

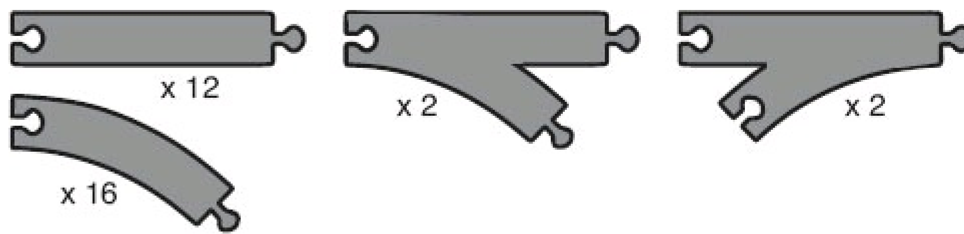


Figura 3.32 Peças de trilhos de madeira de uma ferrovia de brinquedo; cada uma rotulada com o número de cópias do conjunto. Observe que as peças curvadas e em forma de "garfo" ("desvios" ou "pontos") podem ser usadas dos dois lados, desse modo, podem se curvar em ambas as direções, direita ou esquerda. Cada curva subentende 45 graus.

a. Suponha que as peças se encaixam *exatamente* sem nenhuma folga. Forneça uma formulação precisa da tarefa como um problema de busca.

b. Identifique um algoritmo de busca não informada adequada para essa tarefa e justifique a sua escolha.

c. Explique por que a remoção de qualquer um das peças de "forquilha" torna o problema insolúvel.

d. Dê um limite superior do tamanho total do espaço de estados definido pela sua formulação.

(*Dica*: Pense sobre o fator máximo de ramificação para o processo de construção e a profundidade máxima, ignorando o problema de sobreposição de peças e pontas soltas. Comece simulando que cada peça seja única.)

3.17 No final da Seção 3.4.5, mencionamos a **busca de alongamento iterativo**, uma versão iterativa da busca de custo uniforme. A ideia é usar limites crescentes sobre o custo do caminho. Se for gerado um nó cujo custo de caminho exceda o limite atual, ele será imediatamente descartado. Para cada nova iteração, o limite é definido como o menor custo do caminho de qualquer nó descartado na iteração anterior.

a. Mostre que esse algoritmo é ótimo para custos de caminhos em geral.

b. Considere uma árvore uniforme com fator de ramificação b, profundidade de solução d e passos de custo unitário. Quantas iterações exigirá o alongamento iterativo?

c. Agora, considere passos de custos obtidos no intervalo contínuo $[\epsilon, 1]$, onde $0 < \epsilon < 1$. Quantas iterações são exigidas no pior caso?

d. Implemente o algoritmo e aplique-o a instâncias do quebra-cabeça de oito peças e do caixeiro-viajante. Compare o desempenho do algoritmo ao desempenho da busca de custo uniforme e comente seus resultados.

3.18 Descreva um espaço de estados em que a busca de aprofundamento iterativo tenha desempenho muito pior que o da busca em profundidade (por exemplo, $O(n^2)$ *versus* $O(n)$).

3.19 Escreva um programa que receba como entrada duas URLs de páginas da Web e encontre um caminho de links de uma página até a outra. Qual seria uma estratégia de busca apropriada? A busca bidirecional é uma boa ideia? Um mecanismo de busca poderia ser usado para implementar uma função predecessora?

3.20 Considere o problema do mundo do aspirador de pó definido na Figura 2.2.

a. Qual dos algoritmos definidos neste capítulo seria apropriado para este problema? O algoritmo deveria utilizar a busca em árvore ou a busca em grafos?

b. Aplique o algoritmo escolhido para calcular uma sequência ótima de ações para um mundo 3 × 3 que no estado inicial tem sujeira nos três quadrados superiores e o agente está no centro.

c. Construa um agente de busca para o mundo do aspirador de pó e avalie o seu desempenho em um conjunto de mundos 3 × 3 com probabilidade 0,2 de sujeira em cada quadrado. Inclua o custo de busca, bem como o custo do caminho na medida de desempenho, utilizando uma taxa de conversão razoável.

d. Compare o seu melhor agente de busca com um agente reativo aleatório único que aspira se houver sujeira e caso contrário move-se aleatoriamente.

e. Considere o que aconteceria se o mundo fosse ampliado para $n \times n$. Como o desempenho do agente de busca e do agente reativo variam de acordo com n?

3.21 Prove cada uma das afirmações a seguir ou forneça um contraexemplo:

a. A busca em largura é um caso especial de busca de custo uniforme.

b. A busca em profundidade é um caso especial de busca em árvore de melhor escolha.

c. A busca de custo uniforme é um caso especial de uma busca A*.

 **3.22** Compare o desempenho de A* e da RBFS em um conjunto de problemas gerados aleatoriamente do quebra-cabeças de oito peças (com distância de Manhattan) e domínios TSP (com MST — consulte o Exercício 3.30). Discuta os resultados. O que acontece com o desempenho da RBFS quando um pequeno número aleatório é adicionado ao valor heurístico no domínio do quebra-cabeças de oito peças?

3.23 Faça passo a passo a execução de uma busca A* aplicada para o problema de chegar a Bucareste a partir de Lugoj utilizando a distância heurística em linha reta. Isto é, mostre a sequência de nós que o algoritmo vai considerar e a pontuação f, g e h para cada nó.

 **3.24** Imagine um espaço de estado no qual A* utilizando uma BUSCA-EM-GRAFO devolve uma solução subótima com a função h(n) que é admissível mas inconsistente.

3.25 O **algoritmo heurístico de caminho** (Pohl, 1977) é uma busca de melhor escolha em que a função avaliação é $f(n) = (2 - w)g(n) + wh(n)$. Para que valores de w ela é completa? Para que valores é ótima, assumindo que h seja admissível? Que tipo de busca esse algoritmo realiza para $w = 0$, $w = 1$ e $w = 2$?

3.26 Considere a versão ilimitada da grade regular 2-D mostrada na Figura 3.9. O estado inicial está na origem, (0,0), e o estado objetivo está em (x, y).

a. Qual é o fator de ramificação b nesse espaço de estados?

b. Quantos estados distintos existem na profundidade k (para $k > 0$)?

c. Qual é o número máximo de nós expandidos pela busca em largura em árvore?

d. Qual é o número máximo de nós expandidos pela busca em largura em grafos?

e. $h = | u - x | + | v - y |$ é uma heurística admissível para um estado em (u, v)? Explique.

f. Quantos nós são expandidos pela busca em grafos A* utilizando h?

g. h permanece admissível se algumas ligações forem removidas?

h. h permanece admissível se algumas ligações forem adicionadas entre estados não adjacentes?

3.27 n veículos ocupam os quadrados (1,1) através de (n,1) (ou seja, a linha inferior) de uma grade n x n. Os veículos devem se movimentar para a linha superior mas em ordem reversa, assim o veículo i que inicia em (i,1) deve terminar em (n – i + 1,n). Em cada etapa, cada um dos n veículos podem se mover um quadrado para cima, para baixo, à esquerda, à direita ou permanecer no lugar. Outro veículo adjacente (mas não mais que um) pode saltar sobre ele. Dois veículos não podem ocupar o mesmo quadrado.

a. Calcule o tamanho do espaço de estados em função de n.

b. Calcule o fator de ramificação em função de n.

c. Suponha que o veículo i esteja em (x_i, y_i); escreva uma heurística h_i admissível não trivial para a

quantidade de movimentos necessária para chegar ao local objetivo (n – i + 1, n), assumindo não haver outros veículos na grade.

d. Quais das heurísticas seguintes são admissíveis para o problema de mover todos os *n* veículos para seus destinos? Explique.

(i) $\sum_{i=1}^{n} h_i$.

(ii) $\max\{h_1,\ldots, h_n\}$.

(iii) $\min\{h_1,\ldots, h_n\}$.

3.28 Crie uma função heurística para o quebra cabeças de 8 peças que às vezes sobrevaloriza, e mostre como ela pode conduzir a uma solução subótima em uma instância particular. (Se desejar pode utilizar um computador para auxiliá-lo.) Demonstre que se h nunca sobrevalorizar mais que c, A* utilizando h devolve uma solução cujo custo excede por não mais que c o da solução ótima.

3.29 Demonstre que se a heurística for consistente, pode ser admissível. Construa uma heurística admissível que não seja consistente.

3.30 O problema do caixeiro viajante (TSP) pode ser resolvido com a heurística da árvore de geradora mínima (MST – Minimum-Spanning-Tree), que avalia o custo de completar um roteiro de viagem, dado que um roteiro parcial já foi traçado. O custo da MST de um conjunto de cidades é a memor soma do custo de ligação de qualquer árvore que conecte todas as cidades.

a. Mostre como derivar essa heurística de uma versão relaxada do TSP.

b. Mostre que o a heurística da MST domina distâncias retas.

c. Escreva um gerador de problemas para instâncias do TSP onde as cidades são representadas por pontos aleatórios no quadrado de dimensão unitária.

d. Encontre na literatura um algoritmo eficiente para construir a MST e a utilize com a busca A* em grafo para resolver as instâncias do TSP.

3.31 Na Seção 3.6.2, definimos o relaxamento do quebra cabeças de 8 peças no qual uma peça pode mover do quadrado A para o B se B estiver desocupado. A solução exata desse problema define a **heurística de Gaschnig** (Gaschnig, 1979). Explique por que a heurística de Gaschnig é pelo menos tão exata como h1 (peças fora de lugar), e mostre casos em que é mais precisa que ambos h1e h2. (distância de Manhattan). Explique como calcular a heurística de Gaschnig eficientemente.

3.32 Fornecemos duas heurísticas simples para o quebra cabeças de 8 peças: a distância de Manhattan e peças fora de lugar.

Diversas heurísticas na literatura pretendem melhorar isso – consulte, por exemplo Nilson (1971), Mostow e Prieditis (1989) e Hansson et al. (1992). Teste essas asserções implementando as heurísticas e comparando o desempenho dos algoritmos resultantes.

---

[1] Estamos supondo que a maioria dos leitores está na mesma posição, podendo se imaginar com facilidade estar tão desorientado quanto nosso agente. Desculpamo-nos com os leitores romenos que são incapazes de tirar proveito desse exemplo pedagógico.

[2] Muitos tratamentos de resolução de problemas, incluindo edições anteriores deste livro, utilizam a **função sucessor**, que devolve o conjunto de todos os sucessores, em vez das funções AÇÕES e RESULTADO distintas. A função sucessor torna difícil descrever um agente que saiba quais ações pode tentar, mas não as que pode acessar. Além disso, observe que alguns autores utilizam RESULTADO*(a,s)* em vez de RESULTADO(s, a), e alguns usam FAÇA, ou DO em inglês, em vez de RESULTADO.

[3] Esse pressuposto é conveniente algoritmicamente e também teoricamente justificável — vers pág. 649, Capítulo 17.

[4] As implicações de custos negativos são exploradas no Exercício 3.8.

[5] Consulte a Seção 11.2 para definições e algoritmos mais completos.

[6] *NoOp*, ou “não operação”, é o nome de uma instrução em linguagem assembler que não faz nada.

[7] Aqui, e ao longo do livro, o asterisco em C^* significa um valor ótimo de C.

[8] Nossa primeira edição chamava-a de **busca gulosa**; outros autores têm chamado de **busca de melhor escolha**. Nosso uso mais geral do último termo segue Pearl (1984).

[9] Com uma heurística admissível porém inconsistente, A* requer algumas anotações extras para garantir otimalidade.

[10] Na primeira edição desse livro há um esboço rascunhado.

[11] Nos Capítulos 8 e 10, descrevemos as linguagens formais adequadas para essa tarefa. Com descrições formais que podem ser manipuladas, a construção de problemas relaxados poderá ser automatizada. Por enquanto, usaremos o português.

[12] Observe que uma heurística perfeita pode ser obtida simplesmente permitindo que h execute uma busca em largura completa “às cegas”. Assim, deve haver um compromisso entre precisão e tempo de computação para as funções heurísticas.

[13] Ao trabalhar em sentido contrário ao objetivo, o custo da solução exata de cada instância encontrada fica imediatamente disponível. Esse é um exemplo de **programação dinâmica**, que discutiremos mais adiante no Capítulo 17.

CAPÍTULO 4

Além da busca clássica

Ao relaxarmos os pressupostos do capítulo anterior nos aproximamos do mundo real.

O Capítulo 3 se dirigiu a uma única categoria de problemas: observável, determinístico e de ambientes conhecidos, em que a solução é uma sequência de ações. Neste capítulo, examinaremos o que acontece quando esses pressupostos são relaxados. Começaremos com um caso bastante simples: as Seções 4.1 e 4.2 cobrem algoritmos que executam uma **busca local** no espaço de estados, avaliando e modificando um ou mais estados atuais, em vez de explorar sistematicamente os caminhos a partir de um estado inicial. Esses algoritmos são apropriados para problemas em que tudo o que importa é o estado da solução e não o custo do caminho para alcançá-lo. A família de algoritmos de busca local inclui métodos inspirados pela física estatística (**têmpera simulada**) e pela biologia evolutiva (**algoritmos genéticos**).

Assim, nas Seções 4.3 e 4.4, examinaremos o que acontece quando relaxamos os pressupostos do determinismo e da observabilidade. A ideia principal é que, se um agente não pode prever exatamente que percepção vai receber, será necessário considerar o que fazer em cada **contingência** que suas percepções possam revelar. Com a observabilidade parcial, o agente também precisará manter o controle dos estados nos quais ele possa estar.

Finalmente, a Seção 4.5 investiga a **busca on-line** (ou busca e execução), em que o agente se defronta com um espaço de estados que inicialmente é desconhecido e deve ser explorado.

4.1 ALGORITMOS DE BUSCA LOCAL E PROBLEMAS DE OTIMIZAÇÃO

Os algoritmos de busca que vimos até agora foram projetados para explorar sistematicamente espaços de busca. Esse caráter sistemático é alcançado mantendo-se um ou mais caminhos na memória e registrando-se as alternativas que foram exploradas em cada ponto ao longo do caminho e quais delas não foram exploradas. Quando um objetivo é encontrado, o *caminho* até esse objetivo também constitui uma *solução* para o problema.

No entanto, em muitos problemas, o caminho até o objetivo é irrelevante. Por exemplo, no problema das oito rainhas, o que importa é a configuração final das rainhas, e não a ordem em que elas são posicionadas. Essa mesma propriedade geral se mantém para muitas aplicações importantes,

como projeto de circuitos integrados, leiaute de instalações industriais, escalonamento de jornadas de trabalho, programação automática, otimização de rede de telecomunicações, roteamento de veículos e gerenciamento de carteiras.

Se o caminho até o objetivo não importa, podemos considerar uma classe diferente de algoritmos, aqueles que não se preocupam de forma alguma com os caminhos. Os algoritmos de **busca local** operam usando um único **estado atual** (em vez de vários caminhos) e, em geral, se movem apenas para os vizinhos desse estado. Normalmente, os caminhos seguidos pela busca não são guardados. Embora os algoritmos de busca local não sejam sistemáticos, eles têm duas vantagens: (1) usam pouquíssima memória — normalmente um valor constante; e (2) frequentemente podem encontrar soluções razoáveis em grandes ou infinitos (contínuos) espaços de estados para os quais os algoritmos sistemáticos são inadequados.

Além de encontrar objetivos, os algoritmos de busca local são úteis para resolver **problemas de otimização**, nos quais o objetivo é encontrar o melhor estado de acordo com uma **função objetivo**. Muitos problemas de otimização não se adaptam ao modelo de busca "padrão" introduzido no Capítulo 3. Por exemplo, a natureza fornece uma função objetivo — adaptação reprodutiva — que poderia estar tentando otimizar do ponto de vista da evolução de Darwin, mas não existe nenhum "teste de objetivo" e nenhum "custo de caminho" para esse problema.

Para entender a busca local, é muito útil considerar a **topologia de espaço de estados** (como na Figura 4.1). Uma topologia tem ao mesmo tempo "posição" (definida pelo estado) e "elevação" (definida pelo valor da função de custo da heurística ou da função objetivo). Se a elevação corresponder ao custo, o objetivo será encontrar o vale mais baixo — um **mínimo global**; se a elevação corresponder a uma função objetivo, então o objetivo será encontrar o pico mais alto — um **máximo global** (você pode fazer a conversão de um para o outro apenas inserindo um sinal de menos). Os algoritmos de busca local exploram essa topologia. Um algoritmo de busca local **completo** sempre encontra um objetivo, caso ele exista; um algoritmo **ótimo** sempre encontra um mínimo/máximo global.

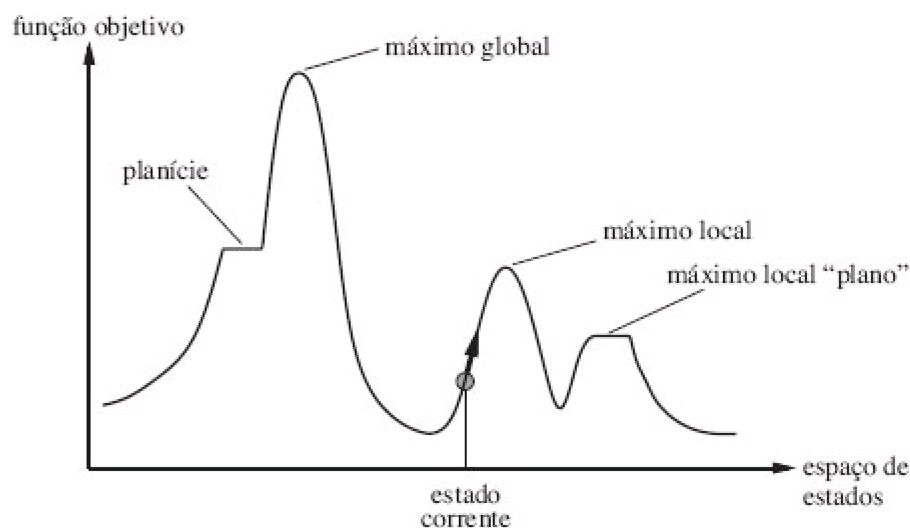


Figura 4.1 Uma topologia de espaço de estados unidimensional, no qual a elevação corresponde à função objetivo. O objetivo é encontrar o máximo global. A busca de subida de encosta modifica o estado atual para tentar melhorá-lo, como mostra a seta. As diversas características topográficas são definidas no texto.

4.1.1 Busca de subida de encosta

O algoritmo de busca de **subida de encosta (**versão **encosta mais íngreme)** é mostrado na Figura 4.2. Ele é simplesmente um laço repetitivo que se move de forma contínua no sentido do valor crescente, isto é, encosta acima. O algoritmo termina quando alcança um "pico" em que nenhum vizinho tem valor mais alto. O algoritmo não mantém uma árvore de busca e, assim, a estrutura de dados do nó atual só precisa registrar o estado e o valor de sua função objetivo. A busca de subida de encosta não examina antecipadamente valores de estados além dos vizinhos imediatos do estado corrente. É como tentar alcançar o cume do Monte Everest em meio a um nevoeiro denso durante uma crise de amnésia.

```
função SUBIDA-DE-ENCOSTA(problema) retorna um estado que é um máximo local
  corrente ← CRIAR-NÓ(ESTADO-INICIAL[problema])
  repita
    vizinho ← um sucessor de corrente com valor mais alto
    se VALOR[vizinho] VALOR[corrente] então retornar ESTADO[corrente]
    corrente ← vizinho
```

Figura 4.2 O algoritmo de busca de subida de encosta é a técnica de busca local mais básica. Em cada passo, o nó atual é substituído pelo melhor vizinho; nessa versão, esse é o vizinho com o VALOR mais alto; porém, se fosse usada uma estimativa de custo de heurística *h*, encontraríamos o vizinho com o *h* mais baixo.

Para ilustrar a subida de encosta, usaremos o **problema das oito rainhas**. Em geral, os algoritmos de busca local utilizam uma **formulação de estados completos**, onde cada estado tem oito rainhas no tabuleiro, uma por coluna. Os sucessores de um estado são todos os estados possíveis gerados pela movimentação de uma única rainha para outro quadrado na mesma coluna (de forma que cada estado tenha $8 \times 7 = 56$ sucessores). A função de custo heurística *h* é o número de pares de rainhas que estão atacando umas às outras, direta ou indiretamente. O mínimo global dessa função é zero, que só ocorre em soluções perfeitas. A Figura 4.3(a) mostra um estado com $h = 17$. A figura também mostra os valores de todos os seus sucessores, sendo que os melhores sucessores têm $h = 12$. Os algoritmos de subida de encosta normalmente fazem uma escolha aleatória entre o conjunto de melhores sucessores, caso exista mais de um.

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

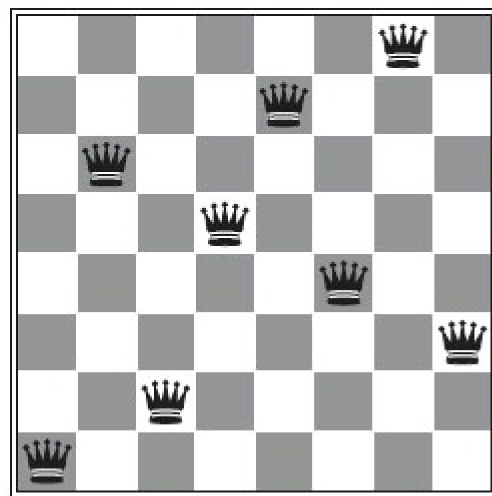


Figura 4.3 (a) Um estado de oito rainhas com estimativa de custo de heurística h = 17, mostrando o valor de h para cada sucessor possível obtido pela movimentação de uma rainha dentro de sua coluna. Os melhores movimentos estão marcados. (b) Um mínimo local no espaço de estados de oito rainhas; o estado tem h = 1, mas todo sucessor tem um custo mais alto.

A subida de encosta às vezes é chamada **busca gulosa local** porque captura um bom estado vizinho sem decidir com antecedência para onde irá em seguida. Embora a gula seja considerada um dos sete pecados capitais, na verdade os algoritmos ambiciosos frequentemente funcionam muito bem. Muitas vezes, a subida de encosta progride com grande rapidez em direção a uma solução porque normalmente é bem fácil melhorar um estado ruim. Por exemplo, a partir do estado da Figura 4.3(a), bastam cinco passos para alcançar o estado da Figura 4.3(b), que tem h = 1 e está muito próxima de uma solução. Infelizmente, a subida de encosta com frequência fica paralisada, pelas seguintes razões:

- **Máximos locais:** um máximo local é um pico mais alto que cada um de seus estados vizinhos, embora seja mais baixo que o máximo global. Os algoritmos de subida de encosta que alcançarem a vizinhança de um máximo local serão deslocados para cima em direção ao pico, mas depois ficarão presos, sem ter para onde ir. A Figura 4.1 ilustra esquematicamente o problema. Em termos mais concretos, o estado da Figura 4.3(b) é de fato um máximo local (isto é, um mínimo local para o custo h); todo movimento de uma única rainha piora a situação.
- **Cordilheiras:** uma cordilheira é mostrada na Figura 4.4. Cordilheiras resultam em uma sequência de máximos locais que torna muito difícil a navegação para algoritmos ambiciosos.

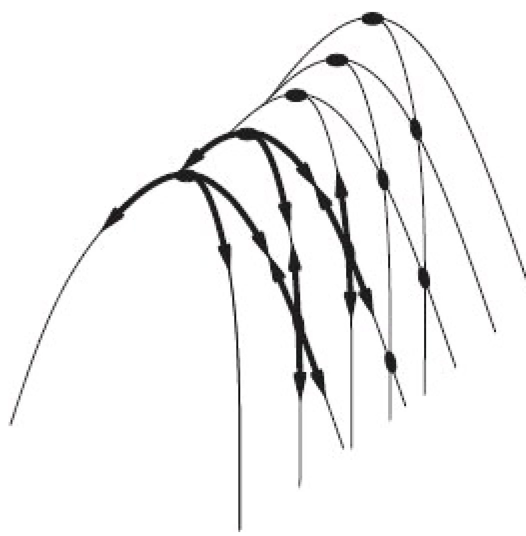


Figura 4.4 Ilustração do motivo pelo qual cordilheiras causam dificuldades na subida de encosta. A malha de estados (círculos escuros) está sobreposta sobre uma cordilheira que se eleva da esquerda para a direita, criando uma sequência de máximos locais que não estão diretamente conectados uns aos outros. A partir de cada máximo local, todas as ações disponíveis apontam encosta abaixo.

- **Platôs:** um platô é uma área plana da topologia de espaço de estados. Ele pode ser um máximo local plano, a partir do qual não existe nenhuma saída encosta acima ou uma **planície**, a partir da qual é possível progredir (veja a Figura 4.1). Uma busca de subida de encosta talvez se perca no platô.

Em cada caso, o algoritmo alcança um ponto em que não há nenhum progresso. A partir de um estado do problema de oito rainhas gerado aleatoriamente, a subida de encosta pela trilha mais íngreme ficará paralisada 86% do tempo, resolvendo apenas 14% de instâncias de problemas. Ela funciona com rapidez, demorando apenas quatro passos em média quando tem sucesso e três quando fica presa — nada mal para um espaço de estados com $8^8 \approx 17$ milhões de estados.

O algoritmo da Figura 4.2 para ao alcançar um platô em que o melhor sucessor tem o mesmo valor do estado corrente. Pode ser uma boa ideia prosseguir — permitir um **movimento lateral**, na esperança de que o platô seja na realidade uma planície, como mostra a Figura 4.1? Normalmente, a resposta é sim, mas devemos ter cuidado. Se sempre permitirmos movimentos laterais quando não houver nenhum movimento encosta acima ocorrerá uma repetição infinita sempre que o algoritmo alcançar um máximo local plano que não seja uma planície. Uma solução comum é impor um limite sobre o número de movimentos laterais consecutivos permitidos. Por exemplo, poderíamos permitir até, digamos, 100 movimentos laterais consecutivos no problema de oito rainhas. Isso aumenta a porcentagem de instâncias de problemas resolvidos por subida de encosta de 14% para 94%. O sucesso tem um custo: o algoritmo demora em média 21 passos aproximadamente para cada instância bem-sucedida e 64 passos para cada falha.

Foram criadas muitas variantes de subida de encosta. A **subida de encosta estocástica** escolhe de formas aleatória os movimentos encosta acima; a probabilidade de seleção pode variar com a declividade do movimento encosta acima. Em geral, isso converge mais lentamente que a subida mais íngreme, mas em algumas topologias de estados encontra soluções melhores. A **subida de encosta pela primeira escolha** implementa a subida de encosta estocástica gerando sucessores ao acaso até ser gerado um sucessor melhor que o estado corrente. Essa é uma boa estratégia quando um estado tem muitos sucessores (por exemplo, milhares).

Os algoritmos de subida de encosta descritos até agora são incompletos — com frequência, eles deixam de encontrar um objetivo que existe porque ficam presos em máximos locais. A **subida de encosta com reinício aleatório** adota o conhecido ditado: "Se não tiver sucesso na primeira vez, continue tentando." Ela conduz uma série de buscas de subida de encosta a partir de estados iniciais[1] gerados de forma aleatória, até encontrar um objetivo. Ela é completa trivialmente, com a probabilidade se aproximando de 1, porque ela irá eventualmente gerar um estado objetivo como estado inicial. Se cada busca de subida de encosta tiver uma probabilidade de sucesso p, o número esperado de reinícios exigidos será $1/p$. Para instâncias das oito rainhas sem permitir movimentos laterais, $p \approx 0{,}14$; assim, precisamos de aproximadamente sete iterações para encontrar um objetivo (seis falhas e um sucesso). O número esperado de passos é o custo de uma iteração bem-sucedida somado a $(1 - p)/p$ vezes o custo de falha ou cerca de 22 passos no total. Quando permitimos movimentos laterais, são necessárias $1/0{,}94 \approx 1{,}06$ iteração em média e $(1 \times 21) + (0{,}06/0{,}94) \times 64 \approx 25$ passos. Então, no caso de oito rainhas, a subida de encosta com reinício aleatório é de fato muito eficiente. Mesmo para três milhões de rainhas, a abordagem pode encontrar soluções em menos de um minuto.[2]

O sucesso da subida de encosta depende muito da forma da topologia do espaço de estados: se houver poucos máximos locais e platôs, a subida de encosta com reinício aleatório encontrará uma boa solução com muita rapidez. Por outro lado, muitos problemas reais têm topologia mais parecida com uma família dispersa de porco-espinhos em um piso plano, com porco-espinho em miniatura vivendo na ponta de cada espinho, *ad infinitum*. Em geral, os problemas NP-difíceis têm um número exponencial de máximos locais em que ficam presos. Apesar disso, um máximo local razoavelmente bom pode ser encontrado com frequência depois de um pequeno número de reinícios.

4.1.2 Têmpera simulada

Um algoritmo de subida de encosta que *nunca* faz movimentos "encosta abaixo" em direção a estados com valor mais baixo (ou de custo mais alto) sem dúvida é incompleto porque pode ficar preso em um máximo local. Em contraste, um percurso puramente aleatório — isto é, mover para um sucessor escolhido uniformemente ao acaso a partir do conjunto de sucessores — é completo, mas extremamente ineficiente. Dessa forma, parece razoável tentar combinar a subida de encosta com um percurso aleatório que resulte de algum modo em eficiência e completeza. A **têmpera simulada** é esse algoritmo. Em metalurgia, a **têmpera** é o processo usado para temperar ou endurecer metais e vidro aquecendo-os a alta temperatura e depois esfriando-os gradualmente, permitindo assim que o material alcance um estado cristalino de baixa energia. Para explicar a têmpera simulada, vamos mudar nosso ponto de vista de subida de encosta para **descida de gradiente** (isto é, minimização do custo) e imaginar a tarefa de colocar uma bola de pingue-pongue na fenda mais profunda em uma superfície acidentada. Se simplesmente deixarmos a bola rolar, ela acabará em um mínimo local. Se agitarmos a superfície, poderemos fazer a bola quicar para fora do mínimo local. O artifício é agitar com força suficiente para fazer a bola sair dos mínimos locais, mas não o bastante para desalojá-la do mínimo global. A solução de têmpera simulada é começar a agitar com força (isto é, em alta temperatura) e depois reduzir gradualmente a intensidade da agitação (ou seja, baixar a temperatura).

O laço de repetição mais interno do algoritmo de têmpera simulada (Figura 4.5) é muito semelhante à subida de encosta. Porém, em vez de escolher o *melhor* movimento, ele escolhe um movimento *aleatório*. Se o movimento melhorar a situação, ele sempre será aceito. Caso contrário, o algoritmo aceitará o movimento com alguma probabilidade menor que 1. A probabilidade decresce exponencialmente com a "má qualidade" do movimento — o valor ΔE segundo o qual a avaliação piora. A probabilidade também decresce à medida que a "temperatura" T se reduz: movimentos "ruins" têm maior probabilidade de serem permitidos no início, quando T estiver alto, e se tornam mais improváveis conforme T diminui. Se o *escalonamento* diminuir T com lentidão suficiente, o algoritmo encontrará um valor ótimo global com probabilidade próxima de 1.

```
função TÊMPERA-SIMULADA(problema, escalonamento) retorna um estado solução
   entradas: problema, um problema
escalonamento, um mapeamento de tempo para "temperatura"
   atual ← CRIAR-NÓ(problema.ESTADO-INICIAL)
   para t = 1 até ∞ faça
      T ← escalonamento[t]
      se T = 0 então retornar corrente
      próximo ← um sucessor de atual selecionado aleatoriamente
      ΔE ← próximo.VALOR – atual.VALOR
      se ΔE > 0 então atual ← próximo
      senão atual ← próximo somente com probabilidade e^(ΔE/T)
```

Figura 4.5 O algoritmo de têmpera simulada, uma versão de subida de encosta estocástica, onde alguns movimentos encosta abaixo são permitidos. Movimentos encosta abaixo são prontamente aceitos no início do escalonamento da têmpera, e depois com menor frequência no decorrer do tempo. A entrada *escalonamento* define o valor da temperatura T como uma função do tempo.

A têmpera simulada foi usada inicialmente de forma extensiva para resolver problemas de leiaute de VLSI no começo dos anos 1980. Ela foi amplamente aplicada ao escalonamento industrial e a outras tarefas de otimização em grande escala. No Exercício 4.4, você será convidado a comparar seu desempenho ao da subida de encosta com reinício aleatório no quebra-cabeça das oito rainhas.

4.1.3 Busca em feixe local

A manutenção de apenas um nó na memória pode parecer uma reação extrema ao problema de limitação de memória. O algoritmo de **busca em feixe local**[3] mantém o controle de k estados, em vez de somente um. Ela começa com k estados gerados aleatoriamente. Em cada passo, são gerados todos os sucessores de todos os k estados. Se qualquer um deles for um objetivo, o algoritmo irá parar. Caso contrário, ele selecionará os k melhores sucessores a partir da lista completa e repetirá o procedimento.

À primeira vista, uma busca em feixe local com k estados talvez pareça não ser nada mais que a execução de k reinícios aleatórios em paralelo, e não em sequência. De fato, os dois algoritmos são

bastante diferentes. Em uma busca com reinício aleatório, cada processo de busca funciona de forma independente dos outros. *Em uma busca em feixe local*, *são repassadas informações úteis entre os k processos paralelos da busca*. Com efeito, os estados que geram os melhores sucessores dizem aos outros: "Venha para cá, aqui está melhor!"O algoritmo logo abandonará as buscas infrutíferas e deslocará seus recursos para o processo em que estiver sendo realizado maior progresso.

Em sua forma mais simples, a busca em feixe local pode se ressentir de uma falta de diversidade entre os k estados — eles podem ficar rapidamente concentrados em uma pequena região do espaço de estados, tornando a busca pouco mais que uma versão dispendiosa de subida de encosta. Uma variante chamada **busca em feixe estocástica**, análoga à subida de encosta estocástica, ajuda a atenuar esse problema. Em vez de escolher o melhor k a partir do conjunto de sucessores candidatos, a busca em feixe estocástica escolhe k sucessores de forma aleatória, com a probabilidade de escolher um determinado sucessor que seja uma função crescente de seu valor. A busca em feixe estocástica guarda alguma semelhança com o processo de seleção natural, pelo qual os "sucessores" (descendência) de um "estado" (organismo) ocupam a próxima geração de acordo com seu "valor" (adaptação ou fitness).

4.1.4 Algoritmos genéticos

Um **algoritmo genético** (ou AG) é uma variante de busca em feixe estocástica na qual os estados sucessores são gerados pela combinação de *dois* estados pais, em vez de serem gerados pela modificação de um único estado. A analogia em relação à seleção natural é a mesma que se dá na busca em feixe estocástica, exceto pelo fato de agora estarmos lidando com a reprodução sexuada, e não com a reprodução assexuada.

Como ocorre com a busca em feixe, os AGs começam com um conjunto de k estados gerados aleatoriamente, chamado **população**. Cada estado, ou **indivíduo**, é representado como uma cadeia sobre um alfabeto finito — muito frequentemente, uma cadeia de valores 0 e 1. Por exemplo, um estado das oito rainhas deve especificar as posições das oito rainhas, cada uma em uma coluna de oito quadrados e, portanto, exigindo $8 \times \log_2 8 = 24$ bits. Como alternativa, o estado poderia ser representado como oito dígitos, cada um no intervalo de 1 a 8 (demonstraremos mais adiante que as duas codificações têm comportamento diferente). A Figura 4.6(a) mostra uma população de quatro cadeias de oito dígitos que representam estados das oito rainhas.

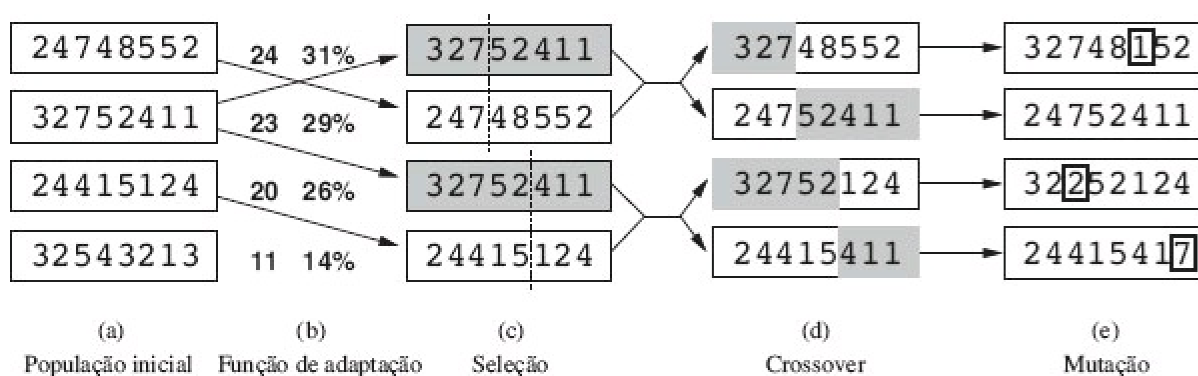


Figura 4.6 O algoritmo genético, ilustrado por sequências de dígitos que representam os estados das

oito rainhas. A população inicial em (a) é classificada pela função de adaptação em (b), resultando em pares de correspondência em (c). Eles produzem descendentes em (d), sujeitos à mutação em (e).

A produção da próxima geração de estados é mostrada na Figura 4.6(b)−(e). Em (b), cada estado é avaliado pela função de avaliação ou (na terminologia do AG) pela **função de adaptação**. Uma função de adaptação deve retornar valores mais altos para estados melhores; assim, para o problema das oito rainhas, usamos o número de pares de rainhas *não atacantes*, que têm o valor 28 para uma solução. Os valores dos quatro estados são 24, 23, 20 e 11. Nessa variante específica do algoritmo genético, a probabilidade de um indivíduo ser escolhido para reprodução é diretamente proporcional à sua pontuação de adaptação, e as porcentagens são mostradas ao lado das pontuações brutas.

Em (c), dois pares escolhidos aleatoriamente são selecionados para reprodução, de acordo com as probabilidades mostradas em (b). Note que um indivíduo é selecionado duas vezes, e um indivíduo não é selecionado de modo algum.[4] Para cada par a ser cruzado, é escolhido ao acaso um ponto de **cruzamento** dentre as posições na cadeia. Na Figura 4.6, os pontos de cruzamento estão depois do terceiro dígito no primeiro par e depois do quinto dígito no segundo par.[5]

Em(d), os próprios descendentes são criados por cruzamento das cadeias pais no ponto de crossover. Por exemplo, o primeiro filho do primeiro par recebe os três primeiros dígitos do primeiro pai e os dígitos restantes do segundo pai, enquanto o segundo filho recebe os três primeiros dígitos do segundo pai e o restante do primeiro pai. Os estados das oito rainhas envolvidos nessa etapa de reprodução são mostrados na Figura 4.7. O exemplo ilustra o fato de que, quando dois estados pais são bastante diferentes, a operação de cruzamento pode produzir um estado que está longe do estado de qualquer pai. Em geral, a população é bastante diversa no início do processo e, assim, o cruzamento (como a têmpera simulada) frequentemente executa grandes passos no espaço de estados bem no início do processo de busca e passos menores mais adiante, quando a maioria dos indivíduos é bastante semelhante.

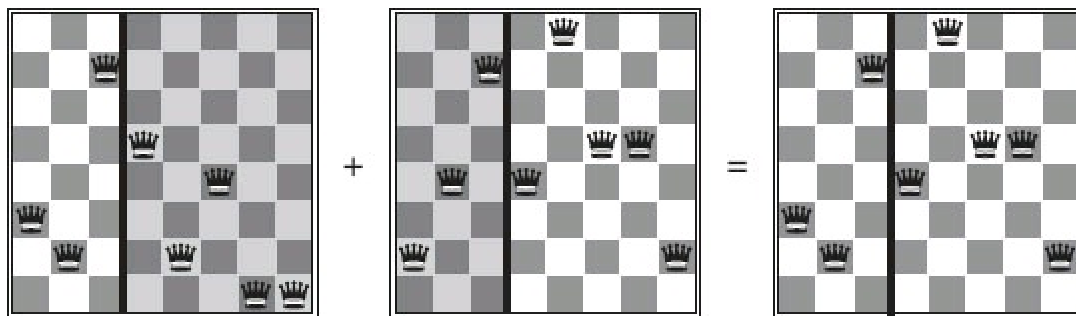


Figura 4.7 Os estados das oito rainhas correspondentes aos dois primeiros pais na Figura 4.6(c) e à primeira descendência da Figura 4.6(d). As colunas sombreadas foram perdidas na etapa de cruzamento, e as colunas não sombreadas foram mantidas.

Finalmente, em (e), cada posição está sujeita à **mutação** aleatória com uma pequena probabilidade independente. Um dígito sofreu mutação no primeiro, no terceiro e no quarto descendente. No problema das oito rainhas, isso corresponde à escolha de uma rainha ao acaso e à movimentação da rainha para um quadrado aleatório em sua coluna. A Figura 4.8 descreve um algoritmo que implementa todas essas etapas.

```
função ALGORITMO-GENÉTICO(população, FN-ADAPTA) retorna um indivíduo
  entradas: população, um conjunto de indivíduos
            FN-ADAPTA, uma função que mede a adaptação de um indivíduo

  repita
    nova_população ← conjunto vazio
    para i = 1 até TAMANHO(população) faça
      x ← SELEÇÃO-ALEATÓRIA(população, FN-ADAPTA)
      y ← SELEÇÃO-ALEATÓRIA(população, FN-ADAPTA)
      filho ← REPRODUZ(x, y)
      se (pequena probabilidade aleatória) então filho ← MUTAÇÃO(filho)
      adicionar filho a nova_população
      população ← nova_população
  até algum indivíduo estar adaptado o suficiente ou até ter decorrido tempo suficiente
  retornar o melhor indivíduo em população, de acordo com FN-ADAPTA
```

```
função REPRODUZ(x, y) retorna um indivíduo
  entradas: x, y, indivíduos pais
  n ← COMPRIMENTO(x)c ← número aleatório de 1 a n
  retornar CONCATENA(SUBCADEIA(x, 1 c), SUBCADEIA(y, c + 1, n))
```

Figura 4.8 Um algoritmo genético.O algoritmo é igual ao que foi representado na Figura 4.6, com uma variação: em sua versão mais popular, cada união de dois pais produz apenas um descendente, e não dois.

Como a busca em feixe estocástico, os algoritmos genéticos combinam uma propensão para subir a encosta com a exploração aleatória e com a troca de informações entre processos de busca paralelos. A principal vantagem dos algoritmos genéticos, se houver, vem da operação de cruzamento. Pode ser demonstrado matematicamente que, se as posições do código genético forem permutadas inicialmente em ordem aleatória, o cruzamento não trará nenhuma vantagem. Intuitivamente, a vantagem vem da habilidade do cruzamento de combinar grandes blocos de genes que evoluem de forma independente para executar funções úteis, elevando assim o nível de granularidade em que a busca opera. Por exemplo, a colocação das três primeiras rainhas nas posições 2, 4 e 6 (em que elas não atacam as outras) constitui um bloco útil que pode ser combinado com outros blocos para elaborar uma solução.

A teoria de algoritmos genéticos explica como isso funciona usando a ideia de **esquema**, uma subcadeia na qual algumas posições podem ser deixadas sem especificação. Por exemplo, o esquema 246***** descreve todos os estados de oito rainhas em que as três primeiras rainhas estão nas posições 2, 4 e 6, respectivamente. As cadeias que correspondem ao esquema (como 24613578) são chamadas **instâncias** do esquema. É possível mostrar que, se o valor de adaptação médio das instâncias de um esquema estiver acima da média, então o número de instâncias do esquema dentro da população crescerá com o passar do tempo. É claro que é improvável que esse efeito seja significativo, caso bits adjacentes estejam totalmente não relacionados uns com os outros porque,

nesse caso, haverá poucos blocos contíguos que proporcionem um benefício consistente. Os algoritmos genéticos funcionam melhor quando os esquemas correspondem a componentes significativos de uma solução. Por exemplo, se a cadeia for uma representação de uma antena, os esquemas poderão representar componentes da antena, como refletores e defletores. É provável que um bom componente seja bom em uma grande variedade de projetos diferentes. Isso sugere que o uso bem-sucedido de algoritmos genéticos exige uma cuidadosa engenharia na representação de estados.

Na prática, os algoritmos genéticos tiveram amplo impacto sobre problemas de otimização, como leiaute de circuitos e escalonamento de prestação de serviços. No momento, não está claro se o interesse pelos algoritmos genéticos surge de seu desempenho ou de suas origens esteticamente atraente na teoria da evolução. Ainda há muito trabalho a ser feito para identificar as condições sob as quais os algoritmos genéticos funcionam bem.

4.2 BUSCA LOCAL EM ESPAÇOS CONTÍNUOS

No Capítulo 2, explicamos a distinção entre ambientes discretos e contínuos, assinalando que a maioria dos ambientes reais é contínua. Ainda assim, nenhum dos algoritmos que descrevemos (exceto a subida de encosta mais íngreme e a de têmpora simulada) pode manipular espaços contínuos e espaços de ação porque têm fatores infinitos de ramificação. Esta seção fornece uma introdução *muito breve* a algumas técnicas de busca local para encontrar soluções ótimas em espaços contínuos. A literatura sobre esse tópico é vasta; muitas técnicas básicas tiveram origem no século XVII, depois do desenvolvimento do cálculo por Newton e Leibniz.[6] Descobriremos usos para essas técnicas em diversos lugares no livro, incluindo os capítulos sobre aprendizado, visão e robótica.

EVOLUÇÃO E BUSCA

A teoria da **evolução** foi desenvolvida por Charles Darwin em *On the Origin of Species By Means of Natural Selection* (1859) e, independentemente, por Alfred Russel Wallace (1858). A ideia central é simples: variações ocorrem na reprodução e serão preservadas em gerações sucessivas em proporção aproximada ao seu efeito sobre a adaptação reprodutiva.

A teoria de Darwin foi desenvolvida sem qualquer conhecimento de como as características dos organismos podem ser herdadas e modificadas. As leis probabilísticas que governam esses processos foram primeiro identificadas por Gregor Mendel (1866), um monge que fez experiências com ervilhas. Muito mais tarde, Watson e Crick (1953) identificaram a estrutura da molécula de DNA e seu alfabeto, AGTC (adenina, guanina, timina, citosina). No modelo-padrão, a variação ocorre por mutações localizadas na sequência de genes e por cruzamento (no qual o DNA de um descendente é gerado pela combinação de longas seções de DNA de cada pai).

A analogia com algoritmos de busca local já foi descrita; a principal diferença entre a busca em feixe estocástico e a evolução é o uso de reprodução sexuada, na qual os sucessores são gerados a partir de *vários* organismos em vez de apenas um. Porém, os mecanismos reais da evolução são muito mais ricos do que permite a maioria dos algoritmos genéticos. Por exemplo, as mutações

podem envolver reversões, duplicações e movimentação de grandes blocos de DNA; alguns vírus tomam emprestado o DNA de um organismo e o inserem em outro; e ainda existem genes de transposição que nada fazem além de copiar a si mesmos muitos milhares de vezes dentro do genoma. Existem até mesmo genes que envenenam células de companheiros potenciais que não transportam o gene, aumentando assim suas chances de replicação. O mais importante é o fato de que os *próprios genes codificam os mecanismos* pelos quais o genoma é reproduzido e convertido em um organismo. Em algoritmos genéticos, esses mecanismos constituem um programa separado que não está representado dentro das cadeias que estão sendo manipuladas.

A evolução de Darwin pode parecer ineficiente, tendo gerado cegamente cerca de 10^{45} organismos sem melhorar uma vírgula sequer suas heurísticas de busca. Contudo, 50 anos antes de Darwin, outro grande naturalista francês chamado Jean Lamarck (1809) propôs uma teoria da evolução pela qual as características adquiridas por adaptação durante a vida de um organismo seriam transmitidas aos seus descendentes. Tal processo seria eficaz, mas não parece ocorrer na natureza. Muito mais tarde, James Baldwin (1896) propôs uma teoria similar em suas características superficiais: que o comportamento aprendido durante a vida de um organismo poderia acelerar a velocidade da evolução. Diferentemente da teoria de Lamarck, a teoria de Baldwin é inteiramente consistente com a evolução de Darwin porque se baseia em forçar a seleção sobre indivíduos que encontram pontos ótimos locais no conjunto de comportamentos possíveis permitidos por sua constituição genética. Simulações em modernos computadores confirmam que o "efeito de Baldwin" é real, desde que a evolução "comum" possa criar organismos cuja medida interna de desempenho esteja de alguma forma correlacionada à adaptação real.

Começaremos com um exemplo. Vamos supor que queiramos instalar três novos aeroportos em qualquer lugar na Romênia, de tal forma que a soma dos quadrados das distâncias de cada cidade no mapa (Figura 3.2) até o aeroporto mais próximo seja minimizada. Então, o espaço de estados é definido pelas coordenadas dos aeroportos: (x_1, y_1), (x_2, y_2) e (x_3, y_3). Esse é um espaço *hexadimensional*; também dizemos que os estados são definidos por seis **variáveis** (em geral, os estados são definidos por um vetor *n*-dimensional de variáveis, **x**). A movimentação nesse espaço corresponde a mover um ou mais dos aeroportos no mapa. A função objetivo $f(x_1, y_1, x_2, y_2, x_3, y_3)$ é relativamente fácil de calcular para qualquer estado específico, uma vez que sejam calculadas as cidades mais próximas. Façamos C*i* o conjunto de cidades mais próximas, cujo aeroporto (no estado atual) é o aeroporto *i*. Então, *na vizinhança do estado atual*, onde os C*i* permanecem constantes, temos:

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2 \,. \tag{4.1}$$

Essa expressão é *localmente* correta, mas não globalmente correta porque os conjuntos C*i* são funções (descontínuas) do estado.

Uma maneira de evitar problemas contínuos é simplesmente **tornar discreta** a vizinhança de cada estado. Por exemplo, podemos mover apenas um aeroporto de cada vez na direção *x* ou *y* por um

valor fixo ±Δ. Com seis variáveis, isso nos dá 12 sucessores para cada estado. Podemos então aplicar qualquer dos algoritmos de busca local descritos anteriormente. Também é possível aplicar diretamente a subida de encosta estocástica e a têmpera simulada, sem tornar o espaço discreto. Esses algoritmos escolhem sucessores aleatoriamente, o que pode ser feito pela geração de vetores aleatórios de comprimento Δ.

Existem muitos métodos que tentam usar o **gradiente** da topologia para encontrar um máximo. O gradiente da função objetivo é um vetor ∇f que fornece a magnitude e a direção da inclinação mais íngreme. Em nosso problema, temos:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right) .$$

Em alguns casos, podemos encontrar um máximo resolvendo a equação $\nabla f = 0$ (por exemplo, isso poderia ser feito se estivéssemos instalando apenas um aeroporto; a solução é a média aritmética das coordenadas de todas as cidades). Porém, em muitos casos, essa equação não pode ser resolvida de forma fechada. Por exemplo, com três aeroportos, a expressão para o gradiente depende das cidades que estão mais próximas a cada aeroporto no estado atual. Isso significa que podemos calcular o gradiente *local*, mas não *global,* por exemplo,

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_i - x_c) . \qquad (4.2)$$

Dada uma expressão localmente correta para o gradiente, podemos realizar uma subida pela encosta mais íngreme, atualizando o estado atual de acordo com a fórmula

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x}),$$

onde α é uma constante pequena chamada frequentemente de **tamanho de passo**. Em outros casos, a função objetivo pode não estar disponível de modo algum em uma forma diferenciável — por exemplo, o valor de um conjunto específico de posições de aeroportos pode ser determinado pela execução de algum pacote de simulação de grande escala. Nesses casos, um **gradiente empírico** pode ser determinado pela avaliação da resposta a pequenos incrementos e decrementos em cada coordenada. A busca de gradiente empírico é igual à subida pela encosta mais íngreme em uma versão do espaço de estados dividida em unidades discretas.

Por trás da frase "α é uma constante pequena" reside uma enorme variedade de métodos para ajuste de α. O problema básico é que, se α é pequeno demais, são necessários muitos passos; se α é grande demais, a busca pode ultrapassar o limite máximo. A técnica de **busca linear** tenta superar esse dilema estendendo a direção de gradiente atual — em geral, pela duplicação repetida de α — até *f* começar a diminuir novamente. O ponto em que isso ocorrer se torna o novo estado atual. Existem diversas escolas de pensamento relacionadas ao modo como a nova direção deve ser escolhida nesse ponto.

Para muitos problemas, o algoritmo mais eficiente é o venerável método de **Newton-Raphson**. Essa é uma técnica geral para encontrar as raízes de funções, isto é, resolver equações da forma $g(x) = 0$. Ela funciona calculando uma nova estimativa para a raiz x de acordo com a fórmula de Newton:

$$x \leftarrow x - g(x)/g'(x).$$

Para encontrar um máximo ou um mínimo de f, precisamos encontrar $\mathbf{x}$ tal que o *gradiente* seja zero (isto é, $\nabla f(\mathbf{x}) = \mathbf{0}$). Desse modo, $g(x)$ na fórmula de Newton se torna $\nabla f(\mathbf{x})$, e a equação de atualização pode ser escrita em forma de vetor de matriz como:

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x})\nabla f(\mathbf{x}),$$

onde $\mathbf{H}f(\mathbf{x})$ é a matriz **hessiana** de segundas derivadas, cujos elementos H_{ij} são dados por $\partial^2 f/\partial x_i \partial x_j$. Para o nosso exemplo do aeroporto, pode-se notar da Equação 4.2 que $\mathbf{H}_f(\mathbf{x})$ é particularmente simples: os elementos fora da diagonal são zero e os elementos da diagonal para o aeroporto i são apenas duas vezes o número de cidades em C_i. Um cálculo rápido mostra que uma etapa da atualização move o aeroporto i diretamente para o centroide de C_i, que é o mínimo da expressão local para f da Equação 4.1.[7] Para problemas de dimensões elevadas, no entanto, calcular n^2 entradas da hessiana e invertê-las pode ser caro, por isso muitas versões aproximadas do método de Newton-Raphson têm sido desenvolvidas.

Os métodos de busca local com máximos locais, cordilheiras e platôs em espaços de estados contínuos, de forma semelhante ao que ocorre em espaços discretos. Reinícios aleatórios e têmpera simulada são recursos que podem ser usados e frequentemente são úteis. Porém, os espaços contínuos de dimensões elevadas são lugares grandes em que é fácil se perder.

Um último tópico sobre o qual seria útil alguma familiarização é a **otimização restrita**. Um problema de otimização é restrito se as soluções devem satisfazer a algumas restrições rígidas sobre os valores de cada variável. Por exemplo, em nosso problema de localização de aeroportos, poderíamos restringir os locais ao interior da Romênia e a áreas de terra firme (e não no meio de lagos). A dificuldade dos problemas de otimização restrita depende da natureza das restrições e da função objetivo. A categoria mais conhecida é a dos problemas de **programação linear**, em que as restrições devem ser desigualdades lineares formando um conjunto *convexo*[8] e a função objetivo também é linear. A complexidade de tempo de programação linear é polinomial no número de variáveis.

A programação linear provavelmente é a classe mais amplamente estudada e de utilidade mais extensa dos problemas de otimização. É um caso especial do problema mais geral de **otimização convexa**, que permite que a região de restrição seja qualquer região convexa e o objetivo seja qualquer função convexa na região de restrição. Sob certas condições, problemas de otimização convexa também são polinomialmente solucionáveis e na prática podem ser viáveis com milhares de variáveis. Vários problemas importantes no aprendizado de máquina e na teoria de controle podem ser formulados como problemas de otimização convexa (ver o Capítulo 20).

4.3 BUSCA COM AÇÕES NÃO DETERMINÍSTICAS

No Capítulo 3, assumimos que o ambiente é totalmente observável e determinístico e que o agente sabe quais são os efeitos de cada ação. Portanto, o agente pode calcular exatamente que estado

resulta de qual sequência de ações e sempre sabe em que estado está. Suas percepções não fornecem nenhuma informação nova após cada ação, embora, evidentemente, informem ao agente o estado inicial.

Quando o ambiente é parcialmente observável ou não determinístico (ou ambos), a percepção torna-se útil. Em um ambiente parcialmente observável, cada percepção ajuda a diminuir o conjunto de estados possíveis onde o agente possa estar tornando assim mais fácil para o agente alcançar seus objetivos. Quando o ambiente é não determinístico, a percepção informa ao agente quais dos resultados possíveis de suas ações realmente ocorreram. Em ambos os casos, a percepção futura não pode ser determinada com antecedência e as ações futuras do agente dependerão daquelas percepções futuras. Então, a solução para um problema não é uma sequência, mas um **plano de contingência** (também conhecido como estratégia) que especifica o que fazer dependendo das percepções recebidas. Nesta seção, examinamos o caso de não determinismo, deixando a observabilidade parcial para a Seção 4.4.

4.3.1 O mundo defeituoso do aspirador de pó

Como exemplo, utilizamos o mundo do aspirador de pó, apresentado pela primeira vez no Capítulo 2 e definido como um problema de busca na Seção 3.2.1. Lembre-se de que o espaço de estados tem oito estados, como mostrado na Figura 4.9. Há três ações — *Esquerda, Direita e Aspirar* – e o objetivo é limpar toda a sujeira (estados 7 e 8). Se o ambiente for observável, determinista e completamente conhecido, o problema é trivialmente solucionável por qualquer um dos algoritmos do Capítulo 3 e a solução é uma sequência de ações. Por exemplo, se o estado inicial for 1, a sequência de ação [*Aspirar*, *Direita*, *Esquerda*] vai alcançar um estado objetivo, 8.

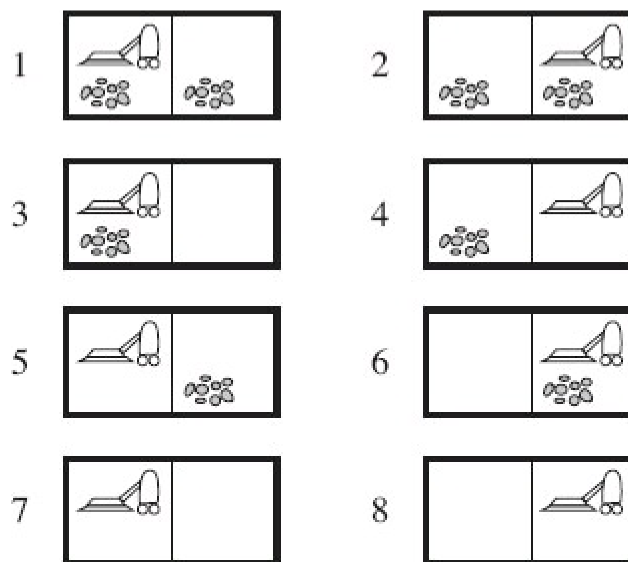


Figura 4.9 Os oito estados possíveis do mundo do aspirador de pó; os estados 7 e 8 são estados objetivo.

Agora, suponha que apresentemos o não determinismo na forma de um aspirador potente, mas defeituoso. No **mundo do aspirador de pó defeituoso**, a ação *aspirar* funciona da seguinte forma:

- Quando aplicada a um quadrado sujo, a ação limpa o quadrado e, por vezes, limpa a sujeira do quadrado adjacente, também.
- Quando aplicado a um quadrado limpo, a ação por vezes deposita sujeira no carpete.[9]

Para fornecer uma formulação precisa desse problema, é preciso generalizar a noção do **modelo de transição** do Capítulo 3. Em vez de definir o modelo de transição por uma função RESULTADO que devolve um único estado, usaremos uma função RESULTADO que devolve um *conjunto* de estados resultantes possíveis. Por exemplo, no mundo do aspirador de pó defeituoso, a ação *Aspirar* no estado 1 leva a um estado no conjunto {5, 7} — a sujeira na área à direita ao lado pode ou não ser aspirada.

Precisamos também generalizar a noção de **solução** para o problema. Por exemplo, se começarmos no estado 1, não existe uma única *sequência* de ações que resolva o problema. Em vez disso, precisaremos de um plano de contingência, como o seguinte:

$$[Aspirar, \textbf{se } Estado = 5, \textbf{então } [Direita, Aspirar] \textbf{ senão } []]. \tag{4.3}$$

Assim, soluções para problemas não determinísticos podem conter comandos **se-então-senão** aninhados, o que significa que elas são *árvores*, e não sequências. Isso permite a seleção de ações com base nas contingências que surgem durante a execução. Muitos problemas no mundo real físico são problemas de contingência, pois a previsão exata é impossível. Por essa razão, muitas pessoas mantêm os olhos bem abertos quando caminham ou dirigem.

4.3.2 Árvores de busca E-OU

A próxima pergunta é como encontrar soluções para os problemas contingentes não determinísticos. Como no Capítulo 3, começamos com a construção de árvores de busca, mas aqui as árvores têm um caráter diferente. Em um ambiente determinístico, a única ramificação é apresentada pelas próprias escolhas do agente em cada estado. Chamamos esses nós de **nós OU**. No mundo do aspirador de pó, por exemplo, em um nó OU o agente escolhe *Esquerda ou Direita ou Aspirar*. Em um ambiente não determinístico, a ramificação é também apresentada pela escolha do resultado do *ambiente* para cada ação. Chamamos esses nós de **nós E**. Por exemplo, a ação *Aspirar* no estado 1 leva a um estado no conjunto {5, 7}, então o agente deverá encontrar um plano para o estado 5 *e* para o estado 7. Esses dois tipos de nós alternados levam a uma **árvore** E-OU, conforme ilustrado na Figura 4.10.

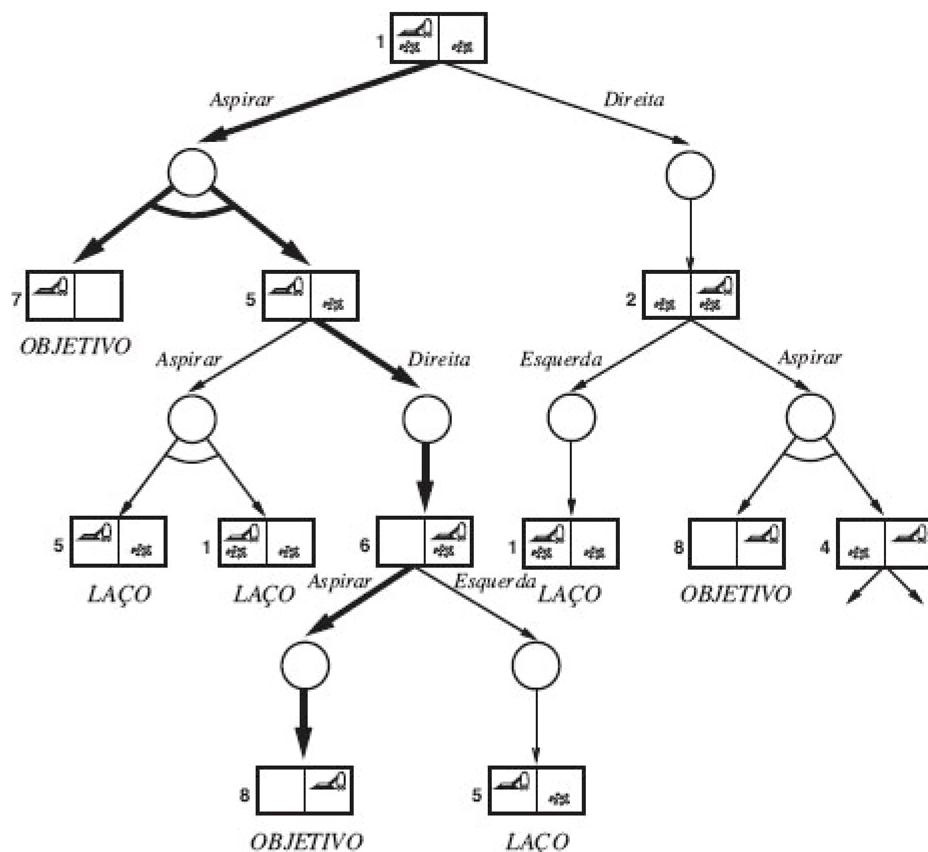


Figura 4.10 Os dois primeiros níveis da árvore de busca para o mundo do aspirador de pó defeituoso. Os nós de estado são os nós OU onde deve ser escolhida alguma ação. Todos os resultados deverão ser tratados nos nós E, mostrados como círculos, como indicado pelo arco ligando os ramos de saída. A solução encontrada é mostrada nas linhas em negrito.

Uma solução para um problema de busca E-OU é uma subárvore que (1) tenha um nó objetivo em cada folha, (2) especifique uma ação em cada um de seus nós OU e (3) inclua todos os ramos resultantes em cada um de seus nós E. A solução é apresentada nas linhas em negrito da figura, que correspondem ao plano dado na Equação 4.3. (O plano usa a notação se-então-senão para tratar com os ramos E, mas quando há mais de dois ramos em um nó pode ser melhor utilizar o comando **case**.)

É simples modificar o agente de resolução de problemas básicos mostrado na Figura 3.1 para executar soluções de contingência desse tipo. Pode-se considerar também um projeto de agente um pouco diferente, em que o agente pode agir *antes* que encontre um plano garantido e trate com algumas contingências apenas à medida que surjam durante a execução. Esse tipo de **intercalação** de busca e execução também é útil para problemas de exploração (ver a Seção 4.5) e para jogos (ver o Capítulo 5).

A Figura 4.11 fornece um algoritmo recursivo em profundidade para busca em grafos E-OU. Um aspecto-chave dos algoritmos é a maneira pela qual ele trata com os ciclos, que surgem frequentemente em problemas não determinísticos (por exemplo, se uma ação, algumas vezes, não tiver efeito ou se um efeito não intencional puder ser corrigido). Se o estado atual for idêntico a um estado no caminho da raiz, ele devolve falha. Isso não significa que *não* haja solução a partir do estado atual, mas significa simplesmente que, se *existe* uma solução não cíclica, ela deve ser acessível a partir da recursão anterior do estado atual, por isso a nova recursão poderá ser descartada. Com essa verificação, podemos garantir que o algoritmo termina em todo espaço de estado finito porque cada caminho deve atingir um objetivo, um beco sem saída ou um estado

repetido. Observe que o algoritmo não verifica se o estado atual é uma repetição de um estado em algum *outro* caminho a partir da raiz, o que é importante para a eficiência do Exercício 4.5, ao investigar essa questão.

função BUSCA-EM-GRAFOS-E-OU(*problema*) **retorna um** *plano condicional ou falha*
  BUSCA-OU(*problema*.Estado-Inicial, *problem*, [ ])

---

função BUSCA-OU(*estado, problema, caminho*) **retorna um** *plano condicional ou falha*
  se *problema*.TESTE-OBJETIVO(*estado*) **então retorna o** plano vazio
  se *estado* estiver *no caminho* **então retorna** *falha*
  para cada ação no *problema*.AÇÕES(*estado*) **faça**
    plano ← BUSCA-E(RESULTADO(*estado, ação*), *problema*, [*estado* | *caminho*])
    se plano ≠ *falha* **então retorna** [*ação* | *plano*]
retorna *falha*

---

função BUSCA-E (*estados, problema, caminho*) **retorna um** *plano condicional ou falha*
  para cada *si* **em** estados **faça**
    plano*i* ← BUSCA-OU(s_i, problema, caminho)
  se o *planoi* ← *falha* **então retorna** *falha*
  retorna [**se** s_1 **então** $plano_1$ **senão se** s_2 **então** $plano_2$ **senão** . . . **se** s_{n-1} **então** $plano_{n-1}$ **senão** *pl*

Figura 4.11 Um algoritmo de busca em grafos E-OU gerado em ambientes não determinísticos. Retorna um plano condicional que atinge um estado objetivo em todas as circunstâncias. (A notação [*x* | *l*] refere-se à lista formada pela adição do objeto *x* à frente da lista *l*.)

Os grafos E-OU podem também ser exploradas por métodos de busca em largura ou da melhor escolha. O conceito de função heurística deve ser modificado para estimar o custo de uma solução de contingência, em vez de em sequência, mas a noção de admissibilidade transfere para depois e existe uma analogia do algoritmo A* para encontrar soluções ótimas. Nas notas bibliográficas, ao final do capítulo, são dadas indicações.

4.3.3 Tente, tente novamente

Considere o mundo do aspirador de pó com incerteza, que é idêntico ao mundo comum (sem defeitos) do aspirador de pó, exceto que as ações de movimento, por vezes falham, deixando o agente no mesmo local. Por exemplo, mover para a *Direita* no estado 1 leva para o conjunto do estado {1,2}. A Figura 4.12 mostra parte da busca em grafos; não há explicitamente mais nenhuma solução acíclica do estado 1, e a BUSCA-EM-GRAFOS-E-OU devolveria falha. Há, no entanto, uma **solução cíclica**, que é continuar tentando para a *Direita* até que funcione. Podemos expressar essa solução pela adição de um **rótulo** para denotar uma parte do plano e usar esse rótulo mais tarde, em vez de repetir o plano em si. Assim, nossa solução cíclica é

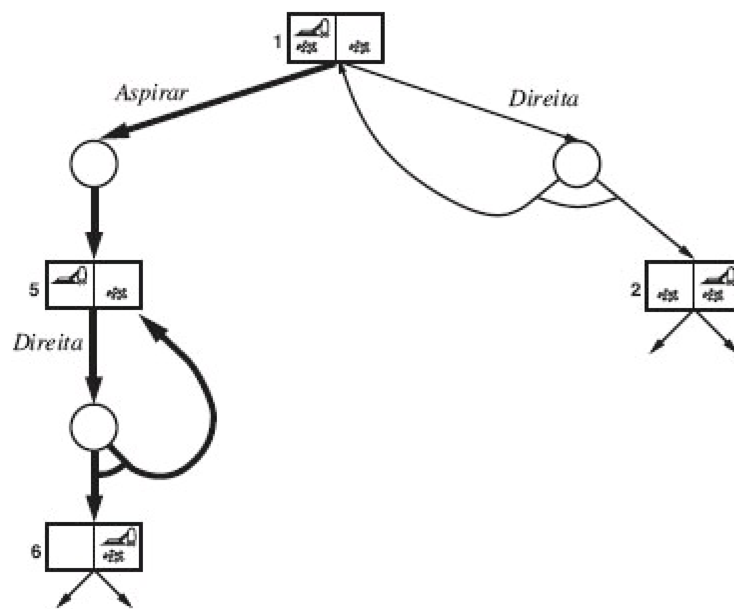


Figura 4.12 Parte da busca em grafos para o mundo do aspirador de pó com incerteza, onde mostramos (alguns) ciclos explicitamente. Todas as soluções para esse problema são planos cíclicos porque não há nenhuma maneira de mover-se de forma confiável.

[*Aspirar* L_1,: *Direita*, **se** *Estado* = 5, **então** L_1 **senão** *Aspirar*].

(A melhor sintaxe para a parte do laço desse plano seria "**enquanto** *estado* = 5 **faça** *Direita*".) Em geral, um plano cíclico pode ser considerado uma solução, desde que cada folha seja um estado objetivo e que uma folha seja acessível de cada ponto no plano. O Exercício 4.6 cobre as modificações necessárias para a BUSCA-EM-GRAFOS-E-OU. A compreensão fundamental é que um laço no espaço de estados que volta a um estado *L* é refletido em um laço no plano que volta ao ponto onde o subplano de estado *L* é executado.

Dada a definição de uma solução cíclica, um agente que executa tal solução eventualmente alcançará o objetivo, *desde que cada resultado de uma ação não determinística ocorra eventualmente*. Essa condição é razoável? Depende do motivo para o não determinismo. Se a ação jogar um dado, então é razoável supor que pode eventualmente resultar um seis. Se a ação for inserir um cartão de chave de hotel na fechadura da porta, mas não funcionar na primeira vez, talvez funcione eventualmente ou talvez seja a chave errada (ou o quarto errado!). Depois de sete ou oito tentativas, a maioria das pessoas vai assumir que o problema é com a chave e vai voltar para a recepção para pedir uma nova. Uma maneira de entender essa decisão é dizer que a formulação do problema inicial (observável, não determinístico) foi abandonada em favor de uma formulação diferente (parcialmente observável, determinístico), onde a falha é atribuída a uma propriedade não observável da chave. Essa questão será abordada novamente no Capítulo 13.

4.4 PESQUISANDO COM OBSERVAÇÕES PARCIAIS

Passaremos agora ao problema de observabilidade parcial, onde a percepção do agente não é suficiente para definir o estado exato. Como observado no início da seção anterior, se o agente estiver em um dos vários estados possíveis, uma ação pode levar a um dos diversos tipos de

resultados possíveis — *mesmo se o ambiente for determinístico*. O conceito fundamental necessário para resolver problemas parcialmente observáveis é o **estado de crença**, que representa a crença atual do agente sobre os possíveis estados físicos em que poderia estar, dada a sequência de ações e percepções até aquele ponto. Começaremos com o cenário mais simples para o estudo dos estados de crença, que é quando o agente não tem sensores; então adicionamos um sensoriamento parcial, bem como ações não determinísticas.

4.4.1 Pesquisar sem observação

Quando as percepções do agente *não fornecem nenhuma informação,* temos o que chamamos problema **sem sensoriamento** ou, algumas vezes, problema **conformante**. Num primeiro momento, pode-se pensar que o agente sem sensoriamento não tem esperança de resolver um problema se não tiver ideia do estado em que ele está; de fato, problemas sem sensoriamento são muitas vezes solúveis. Além disso, os agentes sem sensoriamento podem ser surpreendentemente úteis, principalmente porque eles *não* dependem de sensores funcionando corretamente. Em sistemas de manufatura, por exemplo, foram desenvolvidos muitos métodos engenhosos para orientar corretamente as peças a partir de uma posição inicial desconhecida usando uma sequência de ações completamente sem atividade sensorial. O alto custo da atividade sensorial é outra razão para evitá-la: por exemplo, os médicos geralmente prescrevem um antibiótico de amplo espectro em vez de usar o plano de contingência de fazer um exame de sangue caro, ficar à espera dos resultados e então prescrever um antibiótico mais específico e talvez também a hospitalização devido à progressão da infecção.

Podemos também fazer uma versão sem sensoriamento do mundo do aspirador de pó. Suponha que o agente conheça a geografia do seu mundo, mas não conheça a localização ou a distribuição da sujeira. Nesse caso, seu estado inicial poderia ser qualquer elemento do conjunto {1, 2, 3, 4, 5, 6, 7, 8}. Agora, considere o que acontece se tentar a ação *Direita.* Isso fará com que ele fique em um dos estados {2, 4, 6, 8} — o agente agora tem mais informações! Além disso, a sequência de ação [*Direita, Aspira*] sempre vai acabar em um dos estados {4, 8}. Finalmente, a sequência [*Direita*, *Aspira*, *Esquerda*, *Aspira*] é a garantia de que atingirá o estado objetivo 7, não importa qual seja o estado de início. Dizemos que o agente pode **coagir** o mundo para estado 7.

Para resolver problemas sem sensoriamento, buscamos no espaço do estado de crença em vez de no estado físico.[10] Observe que, no espaço do estado de crença, o problema é *totalmente observável* porque o agente sempre conhece o seu próprio estado de crença. Além disso, a solução (se houver) é sempre uma sequência de ações. Isso porque, como nos problemas comuns do Capítulo 3, as percepções recebidas após cada ação são completamente previsíveis — são sempre vazias! Portanto, não há contingências para planejar. Isso é verdadeiro *mesmo se o ambiente for não determinístico.*

É instrutivo ver como é construído o problema de busca de estado de crença. Suponha que o problema físico subjacente *P* seja definido por AÇÕES_P, RESULTADO_P, TESTAR-OBJETIVO_P e Custo do passo_P. Então, podemos definir o problema sem sensoriamento correspondente da seguinte forma:

- **Estados de crença**: O espaço de estado de crença inteiro contém cada conjunto possível de estados físicos. Se P tiver N estados, então o problema sem sensoriamento terá até 2^n estados, embora muitos possam estar inacessíveis a partir do estado inicial.
- **Estado inicial**: Normalmente, o conjunto de todos os estados em P, embora em alguns casos o agente tenha mais conhecimento do que isso.
- **Ações**: Isso é um pouco complicado. Suponha que o agente esteja no estado de crença $b = \{s_1, s_2\}$, mas as $\text{AÇÕES}_P(s_1) \neq \text{AÇÕES}_P(s_2)$; então, o agente não tem certeza de quais ações são aplicáveis. Se assumirmos que as ações não aplicáveis não têm nenhum efeito sobre o meio ambiente, então é seguro considerar a *união* de todas as ações em qualquer dos estados físicos no estado de crença atual b:

$$\text{AÇÕES}(b) = \bigcup_{s \in b} \text{AÇÕES}_P(s).$$

  Por outro lado, se uma ação não aplicável pode ser o fim do mundo, é mais seguro permitir apenas a *interseção*, isto é, o conjunto de ações aplicáveis em *todos* os estados. Para o mundo do aspirador de pó, cada estado tem as mesmas ações aplicáveis; assim, ambos os métodos dão o mesmo resultado.
- **Modelo de transição**: O agente não sabe qual estado no estado de crença é o correto, por isso, ele só sabe que pode chegar a qualquer um dos estados resultantes da aplicação da ação para um dos estados físicos no estado de crença. Para ações determinísticas, o conjunto de estados que pode ser alcançado é

$$b' = \text{RESULTADO}(b, a) = \{s' : s' = \text{RESULTADO}_P(s, a) \text{ e } s \in b\}. \quad (4.4)$$

  Com ações determinísticas, b' nunca será maior do que b. Com não determinísticas, temos

$$\begin{aligned} b' = \text{RESULTADO}(b, a) &= \{s' : s' \in \text{RESULTADOS}_P(s, a) \text{ e } s \in b\} \\ &= \bigcup_{s \in b} \text{RESULTADO}_P(s, a), \end{aligned}$$

  que pode ser maior do que b, como mostrado na Figura 4.13. O processo de geração de um novo estado de crença após a ação é chamado de etapa de **predição**; a notação $b' = \text{PREDIÇÃO}_P(b, a)$ é mais apropriada.
- **Testar objetivo**: O agente quer um plano que é certo que funcione, o que significa que um estado de crença satisfaz o objetivo somente se *todos* os estados físicos nele satisfizerem TESTE-OBJETIVO_P. O agente poderá atingir *acidentalmente* o objetivo antes, mas não saberá que fez isso.
- **Custo do passo**: Esse é também complicado. Se a mesma ação pode ter custos diferentes em diferentes estados, então o custo de tomar uma ação em um dado estado de crença pode ser um de vários valores (isso dá origem a uma nova classe de problemas, que vamos explorar no Exercício 4.9). Por ora assumimos que o custo de uma ação é o mesmo em todos os estados e por isso pode ser transferido diretamente do problema físico subjacente.

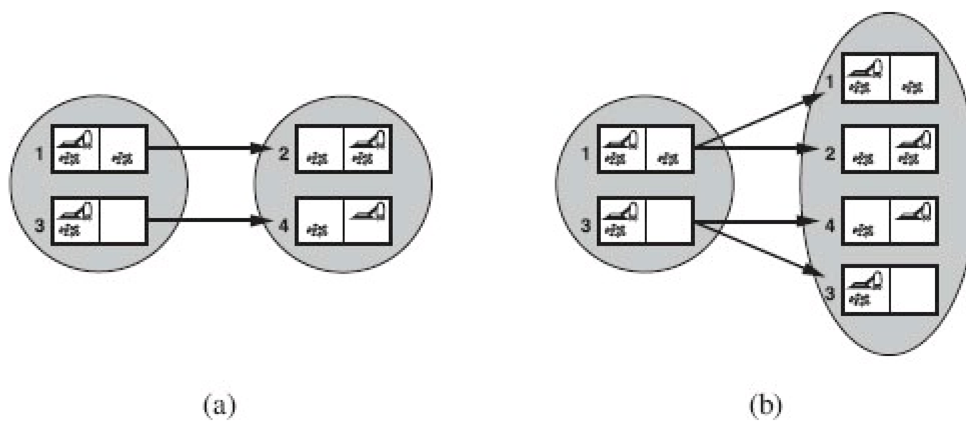


Figura 4.13 (a) Previsão do próximo estado de crença para o mundo do aspirador de pó sem sensoriamento com uma ação determinística, *Direita*. (b) Previsão para o mesmo estado de crença e ação na versão incerta do mundo do aspirador de pó sem sensoriamento.

A Figura 4.14 mostra o espaço de estado de crença acessível para o mundo do aspirador de pó determinístico, sem sensoriamento. Existem apenas 12 estados de crença acessíveis de $2^8 = 256$ estados de crença possíveis.

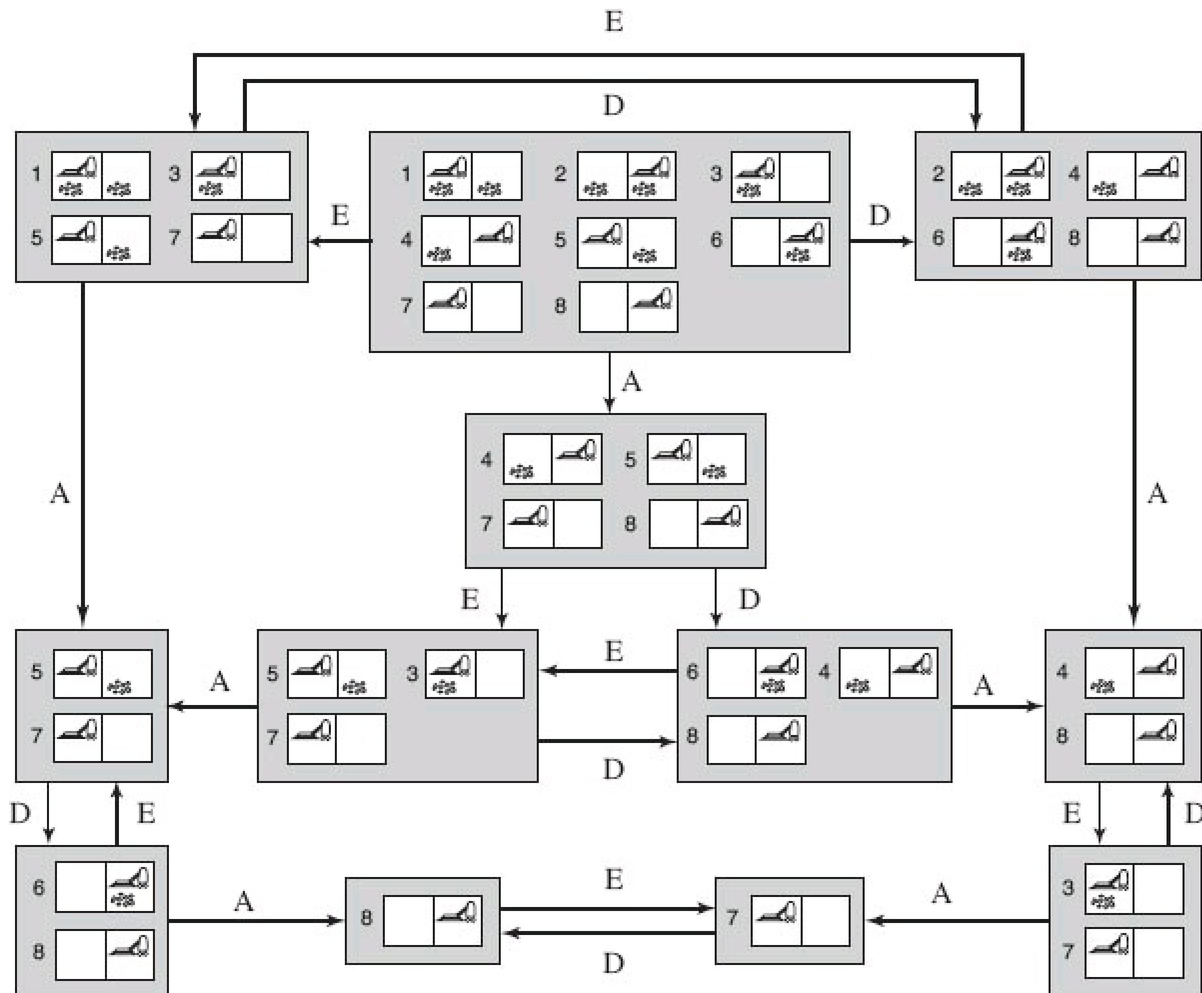


Figura 4.14 A porção acessível do espaço de estado de crença para o mundo determinístico do aspirador de pó, sem sensoriamento. Cada caixa sombreada corresponde a um estado de crença individual. Em um passo qualquer, o agente está em um estado de crença particular, mas não sabe em que estado físico está. O estado de crença inicial (completa ignorância) é a caixa central superior. As ações são representadas pelas ligações rotuladas. Para maior clareza, os laços para um mesmo estado de crença são omitidos.

As definições anteriores permitem a construção automática da formulação do problema de estado de crença a partir da definição do problema físico subjacente. Uma vez feito isso, podemos aplicar qualquer um dos algoritmos de busca do Capítulo 3. Na verdade, podemos fazer um pouco mais do que isso. Na busca em grafos "comum", os estados recentemente gerados são testados para verificar se são idênticos aos estados existentes. Isso funciona também para os estados de crença; por exemplo, na Figura 4.14, a sequência de ação [*Aspirar*, *Esquerda*, *Aspirar*] inicia quando o estado inicial atinge o mesmo estado de crença [*Direita*, *Esquerda*, *Aspirar*], ou seja, {5, 7}. Agora, considere o estado de crença alcançado por [*Esquerda*], ou seja, {1, 3, 5, 7}. Obviamente, não é idêntico a {5, 7}, mas é um *superconjunto*. É fácil provar (Exercício 4.8) que, se uma sequência de ação é uma solução para um estado de crença *b*, é também uma solução para qualquer subconjunto de *b*. Assim, podemos descartar alcançar um caminho {1, 3, 5, 7} se {5, 7} já foi gerado. Por outro lado, se {1, 3, 5, 7} já foi gerado e verificado que é solucionável, é garantido que qualquer *subconjunto,* como {5, 7}, é solucionável. Esse nível extra de poda pode melhorar drasticamente a eficiência da resolução do problema sem sensoriamento.

No entanto, mesmo com essa melhora, a resolução do problema sem sensoriamento como descrito, raramente é viável na prática. A dificuldade não é tanto a vastidão do espaço do estado de crença — mesmo sendo exponencialmente maior do que o espaço de estado físico subjacente; na maioria dos casos, o fator de ramificação e a extensão da solução no espaço do estado de crença e no espaço do estado físico não são tão diferentes. A verdadeira dificuldade reside no tamanho de cada estado de crença. Por exemplo, o estado de crença inicial para o mundo do aspirador de pó 10×10 contém 100×2^{100} ou cerca de 10^{32} estados físicos — é demasiado se utilizarmos a representação atômica, que é uma lista explícita de estados.

Uma solução é utilizar alguma descrição mais compacta para representar o estado de crença. Em português, poderíamos dizer que o agente não sabe "nada" no estado inicial; após mover-se para a *Esquerda*, poderíamos dizer "Não está na coluna mais à direita", e assim por diante. O Capítulo 7 explica como fazer isso em um esquema de representação formal. Outra abordagem é evitar os algoritmos de busca-padrão, que tratam o estado de crença como caixas-pretas, tal como qualquer outro problema de estado. Em vez disso, podemos olhar no *interior* dos estados de crença e desenvolver algoritmos de **busca de estado de crença incremental** que constroem a solução de um estado físico de cada vez. Por exemplo, no mundo do aspirador de pó sem sensoriamento, o estado de crença inicial é {1, 2, 3, 4, 5, 6, 7, 8} e temos que encontrar uma sequência de ação que funcione em todos os oito estados. Podemos fazer isso encontrando primeiro uma solução que funcione para o estado 1; então verificamos se funciona para o estado 2; senão, voltamos e encontramos uma solução diferente para o estado 1, e assim por diante. Como uma busca E-OU tem que encontrar uma solução para todos os ramos de um nó E, esse algoritmo tem que encontrar uma solução para cada estado no estado de crença; a diferença é que a busca E-OU pode encontrar uma solução diferente para cada ramo, enquanto uma busca de estado de crença incremental tem que encontrar *uma* solução que funcione para *todos* os estados.

A principal vantagem da abordagem incremental é que normalmente é capaz de detectar a falha rapidamente — quando um estado de crença é insolúvel, geralmente é o caso dos primeiros estados examinados que consistem de um pequeno subconjunto do estado de crença, também é insolúvel. Em alguns casos, isso leva a um aumento de velocidade proporcional ao tamanho dos estados de crença,

que poderá ser tão grande quanto o espaço de estado físico em si.

Mesmo o algoritmo de solução mais eficiente não é de muita utilidade quando não existem soluções. Muitas coisas simplesmente não podem ser feitas sem atividade sensorial. Por exemplo, o quebra-cabeça de oito peças sem sensoriamento é impossível. Por outro lado, um pouco de atividade sensorial pode contribuir bastante. Por exemplo, cada instância do quebra-cabeça de oito peças pode ser resolvido se apenas uma área for perceptível — a solução envolve mover cada peça, por vez, para um local perceptível e, então, se manter atualizado da sua localização de acordo com as ações tomadas.

4.4.2 Busca com observações

Para um problema geral parcialmente observável, temos que especificar como o ambiente gera percepções para o agente. Por exemplo, poderíamos definir a atividade sensorial local do mundo do aspirador de pó como sendo aquela em que o agente tem um sensor de posição e um sensor de sujeira local, mas não tem um sensor capaz de detectar a sujeira em outros quadrados. A especificação do problema formal inclui uma função PERCEPÇÃO(s) que devolva a percepção recebida em um determinado estado (se a atividade sensorial for não determinística, então usamos uma função PERCEPÇÃO que devolva um conjunto de percepções possíveis). Por exemplo, na atividade sensorial local do mundo do aspirador de pó, a PERCEPÇÃO no estado 1 é [*A*, *Sujo*]. Problemas totalmente observáveis são um caso especial em que a PERCEPÇÃO(s) $= s$ para cada estado s, enquanto os problemas sem sensoriamento são um caso especial em que a PERCEPÇÃO(s) = *nulo*.

Quando as observações são parciais, normalmente é o caso em que vários estados podem produzir qualquer percepção determinada. Por exemplo, a percepção [*A*, *Sujo*] é produzida pelo estado 3, bem como pelo estado 1. Assim, sendo essa a percepção inicial, o estado de crença inicial para a atividade sensorial local do mundo do aspirador de pó será {1, 3}. As AÇÕES, CUSTO-DO-PASSO e TESTE-OBJETIVO são construídas a partir do problema físico subjacente, assim como os problemas sem sensoriamento, mas o modelo de transição é um pouco mais complicado. Podemos pensar em transições de um estado de crença para o próximo de uma determinada ação como ocorrendo em três etapas, conforme mostra a Figura 4.15:

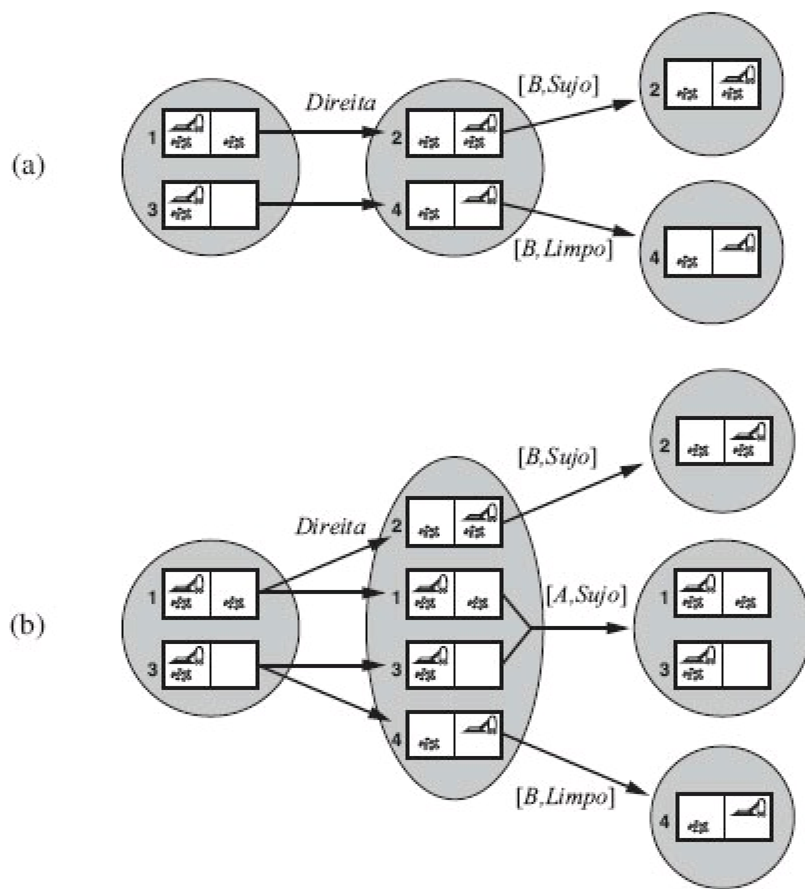


Figure 4.15 Dois exemplos de transições dos mundos do aspirador de pó com sensoriamento local. (a) No mundo determinístico, *Direita* é aplicada no estado de crença inicial, resultando em um estado de crença novo com dois estados físicos possíveis; para aqueles estados, as pecepções possíveis são [*B*, *Sujo*] e [*B*, *Limpo*], levando a dois estados de crença, cada um dos quais unitário. (b) No mundo com observação local de sujeira, *Direita* é aplicada no estado de crença inicial, originando um novo estado de crença com quatro estados físicos; para aqueles estados, as percepções possíveis são [*A*, *Sujo*], [*B*, *Sujo*], e [*B*, *Limpo*], levando a três estados de crença, conforme mostrado.

- A fase de **previsão** é a mesma dos problemas sem sensoriamento: dada uma ação *a* no estado de crença *b*, o estado de crença previsto é $\hat{b} = \text{PREDIÇÃO}(b, a)$.[11]
- A fase de **previsão de observação** determina o conjunto de percepções *o* que poderia ser observado no estado de crença previsto:

$$\text{PERCEPÇÕES-POSSÍVEIS}(\hat{b}) = \{o: o = \text{PERCEPÇÃO}(s) \text{ e } s \in \hat{b}\}.$$

- A fase de **atualização** determina, para cada percepção possível, o estado de crença que resultaria da percepção. O novo estado de crença b_o é apenas o conjunto de estados em $\hat{b}$ que poderia ter produzido a percepção:

$$b_o = \text{ATUALIZAÇÃO}(\hat{b}, o) = \{s: o = \text{PERCEPÇÃO}(s) \text{ e } s \in \hat{b}\}.$$

Observe que cada estado de crença b_o atualizado não pode ser maior do que o estado de crença previsto $\hat{b}$; as observações podem apenas ajudar a reduzir a incerteza comparado ao caso sem sensoriamento. Ademais, para o sensoriamento determinístico, os estados de crença para as

diferentes percepções possíveis serão disjuntos, formando uma *partição* do estado de crença original previsto.

Colocando esses três estágios juntos, obtemos os estados de crença possíveis resultantes de determinada ação e as percepções subsequentes possíveis:

$$\text{RESULTADOS}(b, a) = \{b_o : b_o = \text{ATUALIZAÇÃO}(\text{PREDIÇÃO}(b, a), o) \text{ e } o \in \text{PERCEPÇÕES-POSSÍVEIS}(\text{PREDIÇÃO}(b, a))\}. \quad (4.5)$$

Novamente, o não determinismo do problema parcialmente observável vem da incapacidade de predizer exatamente que percepções serão recebidas após a ação; o não determinismo subjacente no ambiente físico pode *contribuir* para essa incapacidade, ampliando o estado de crença na fase da predição, levando a mais percepções no estágio de observação.

4.4.3 Resolvendo problemas parcialmente observáveis

A seção anterior mostrou como derivar a função RESULTADOS para um problema de estado de crença não determinístico de um problema físico subjacente e a função PERCEPÇÃO. Dada tal formulação, o algoritmo de busca E-OU da Figura 4.11 pode ser aplicado diretamente para obter uma solução. A Figura 4.16 mostra parte da árvore de busca para o mundo do aspirador de pó de sensoriamento local, assumindo uma percepção inicial [*A*, *Sujo*]. A solução é o plano condicional

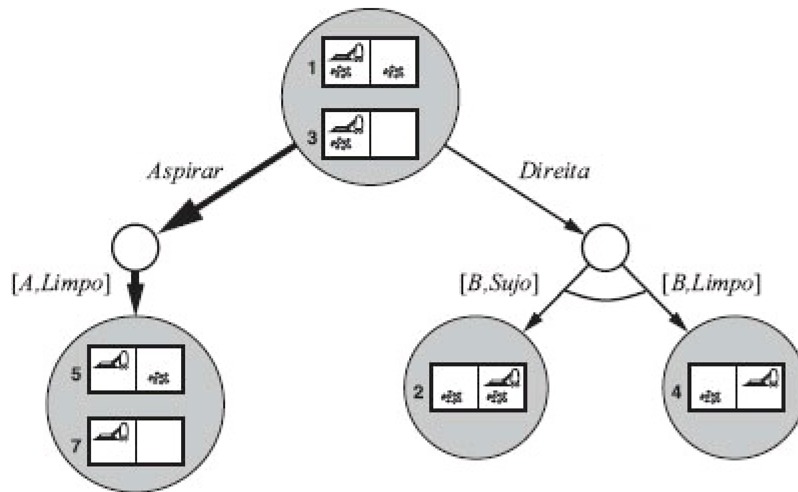


Figura 4.16 O primeiro nível da árvore de busca E-OU de um problema do mundo do aspirador de pó de sensoriamento local; *Aspirar* é o primeiro passo da solução.

[*Aspirar*, *Direita*, **se** *estadoB* = {6}, **então**, *Aspirar* **senão** [ ]].

Observe que, por termos fornecido um problema de estado de crença para o algoritmo de busca E-OU, ele devolveu um plano condicional que testa o estado de crença em vez do estado real. Deveria ser assim mesmo: em um ambiente parcialmente observável, o agente não será capaz de executar uma solução que requeira testar o estado real.

Como no caso dos algoritmos de busca-padrão aplicados a problemas sem sensoriamento, o algoritmo de busca E-OU trata os estados de crença como caixas-pretas, assim como quaisquer

outros estados. Pode-se melhorar essa situação verificando os estados de crença gerados anteriormente que são subconjuntos ou superconjuntos do estado atual, assim como os problemas sem sensoriamento. Pode-se também derivar algoritmos de busca incremental, análogos àqueles descritos para problemas sem sensoriamento, que fornecem avanço substancial sobre a abordagem da caixa-preta.

4.4.4 Um agente para ambientes parcialmente observáveis

O projeto de um agente de resolução de problema para ambientes parcialmente observáveis é bastante semelhante ao do agente de resolução de problemas simples da Figura 3.1: o agente formula um problema, chama um algoritmo de busca (como o de BUSCAE-EM-GRAFOS-E-OU) para resolvê-lo e executa a solução. Há duas diferenças principais. Primeiro, a solução para um problema será um plano condicional, em vez de uma sequência; se o primeiro passo for uma expressão *se-então-senão*, o agente terá que testar a condição da parte *se* e executar a parte *então* ou a parte *senão* conforme o caso. Segundo, o agente terá que manter o seu estado de crença, enquanto executa ações e recebe percepções. Esse processo se assemelha ao processo de atualização da previsão de observação da Equação 4.5, mas na verdade é mais simples porque a percepção é dada pelo ambiente, e não calculada pelo agente. Dado um estado de crença inicial *b*, uma ação *a* e uma percepção *o*, o novo estado de crença será:

$$b' = \text{ATUALIZAÇÃO}\ (\text{PREDIÇÃO}(b, a), o). \qquad (4.6)$$

A Figura 4.17 mostra o estado de crença sendo mantido no mundo do aspirador de pó do *jardim de infância* com sensoriamento local, onde qualquer área pode ficar suja, a qualquer momento, a menos que o agente esteja naquele momento limpando-a ativamente.[12]

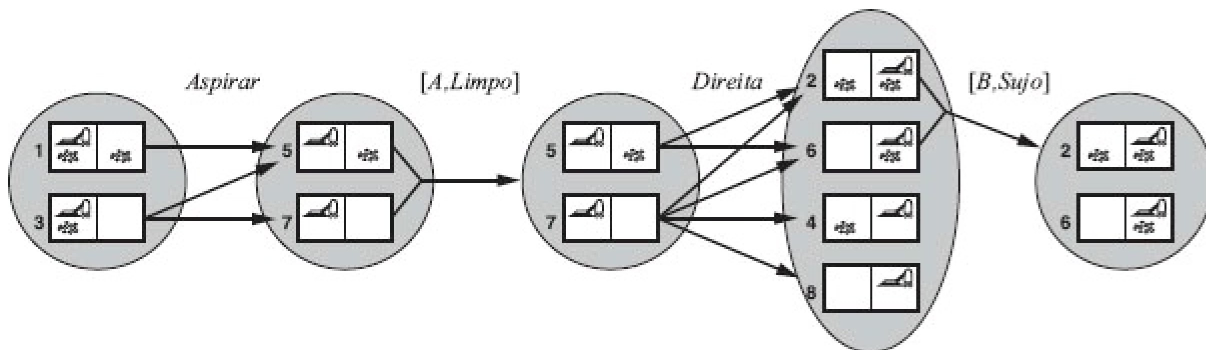
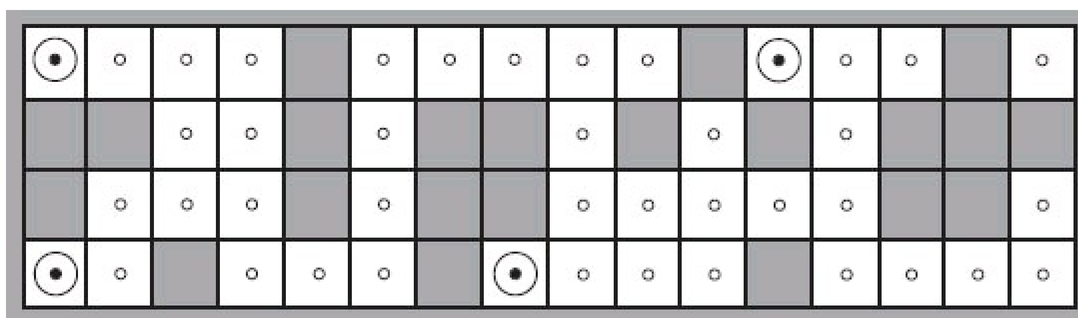


Figura 4.17 Dois ciclos de previsão-atualização da manutenção do estado de crença do mundo do aspirador de pó de um jardim de infância com sensoriamento local.

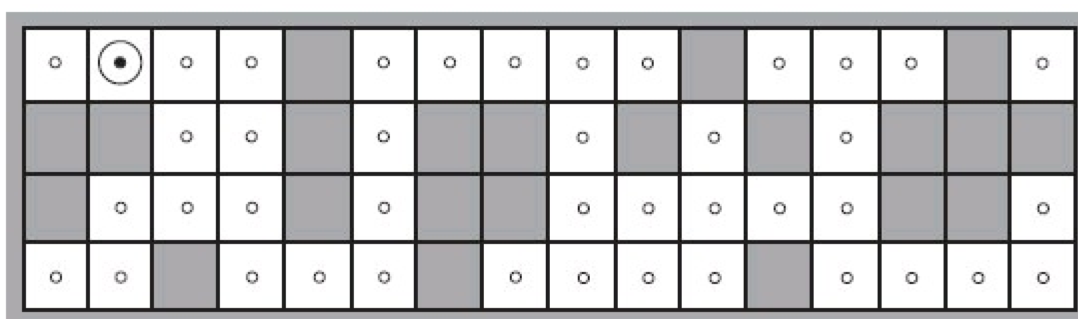
Em ambientes parcialmente observáveis, que incluem a grande maioria dos ambientes do mundo real, manter um estado de crença é uma função essencial de qualquer sistema inteligente. Essa função possui vários nomes, incluindo **monitoramento**, **filtragem** e **estimativa de estado**. A Equação 4.6 é chamada de **estimador de estado recursivo** porque calcula o estado da nova crença a partir da anterior em vez de examinar a sequência inteira de percepção. Se o agente não "ficar para trás", o cálculo tem que acontecer tão rápido quanto as percepções estão acontecendo. Na medida que o

ambiente se torna mais complexo, o cálculo da atualização exata se torna inviável, e o agente terá que calcular um estado de crença aproximado, talvez enfocando as implicações da percepção dos aspectos do ambiente que são de interesse atual. A maioria dos trabalhos sobre esse problema tem sido feita para ambientes de estado contínuo e estocásticos com as ferramentas da teoria da probabilidade, conforme será explicado no Capítulo 15. Aqui vamos mostrar um exemplo em ambiente discreto com sensores determinísticos e ações não determinísticas.

O exemplo diz respeito a um robô com a tarefa de **localização**, resolvendo o problema onde estiver, dado um mapa do mundo e uma sequência de percepções e ações. Nosso robô será colocado no ambiente de labirinto da Figura 4.18. O robô é equipado com quatro sonares sonar que informam a existência de um obstáculo, a parede exterior ou um quadrado preto na figura — em cada uma das quatro direções da bússola. Assumimos que os sensores fornecem dados perfeitamente corretos e que o robô tem um mapa correto do ambiente. Mas, infelizmente, o sistema de navegação do robô está quebrado; então, quando ele executa uma ação de *movimento*, move-se aleatoriamente a um dos quadrados adjacentes. A tarefa do robô é determinar a sua localização atual.



(a) Posições possíveis do robô após E_1 = NSO



(b) Posições possíveis do robô após E_1 = NSO, E_2 = NS

Figure 4.18 Posições possíveis do robô, ⊙, (a), após uma observação $E_1 = NSO$ e (b) após uma segunda observação $E_2 = NS$. Quando os sensores precisos (sem ruído) e o modelo de transição é preciso, não existem outras localizações possíveis para o robô de acordo com essa sequência de duas observações.

Suponha que o robô acabe de ser ligado, por isso não sabe onde está. Assim, sua crença de estado inicial *b* consiste no conjunto de todas as localidades. O robô recebe a percepção *NSO*, o que significa que existem obstáculos ao norte, oeste e sul, e faz uma atualização utilizando a equação b_o = ATUALIZAÇÃO(b), gerando as quatro localizações mostradas na Figura 4.18(a). Você pode inspecionar o labirinto para verificar que aquelas são as únicas quatro localizações que produzem a percepção *NSO*.

Em seguida, o robô executa uma ação de *movimento*, mas o resultado é não determinístico. A nova crença de estado, b_a = PREDIÇÃO(b_o, *Move*), contém todos os locais que estão a um passo dos locais em b_o. Quando a segunda percepção, *NS*, chega, o robô ATUALIZA(b_a, *NS*) e descobre que o estado de crença foi reduzida a uma única localização mostrada na Figura 4.18(b). Essa é a única localização que poderia ser o resultado de

$$\text{ATUALIZAÇÃO (PREDIÇÃO(ATUALIZAÇÃO}(b, NSO), Move), NS).$$

Com ações não deterministícas, a etapa PREDIÇÃO aumenta o estado de crença, mas a etapa ATUALIZAÇÃO novamente o reduz na medida em que a percepção fornece alguma informação útil de identificação. Às vezes, a percepção não ajuda muito para a localização: se houver um ou mais corredores longos leste-oeste, um robô poderá receber uma longa sequência de percepções *NS*, mas nunca saberá em que corredor está.

4.5 AGENTES DE BUSCA ON-LINE EM AMBIENTES DESCONHECIDOS

Até agora nos concentramos em agentes que utilizam algoritmos de **busca off-line**. Eles calculam uma solução completa antes de entrar no mundo real e depois executam a solução sem recorrer a suas percepções. Em contraste, um agente de **busca on-line**[13] **intercala** computação e ação: primeiro, ele executa uma ação, depois observa o ambiente e calcula a próxima ação. A busca on-line é uma boa ideia em domínios dinâmicos ou semidinâmicos — domínios em que existe uma penalidade por continuar calculando durante muito tempo. A busca on-line é também útil em domínios não determinísticos porque permite que o agente concentre seus esforços computacionais sobre as contingências que realmente surgem em vez das que *poderiam* acontecer, mas provavelmente não ocorrerão. Claro, há uma compensação: quanto mais um agente planejar o futuro, menos vezes irá encontrar-se em uma posição difícil.

A busca on-line é uma ideia *necessária* para ambientes desconhecidos, onde o agente não conhece quais estados existem ou o que suas ações fazem. Nesse estado de ignorância, o agente enfrenta um **problema de exploração** e deve usar suas ações como experimentos, a fim de aprender o suficiente para fazer a deliberação (predição e atualização) valer a pena.

O exemplo canônico de busca on-line é um robô colocado em um novo edifício e que tem de explorá-lo para elaborar um mapa que possa ser usado com a finalidade de ir de *A* até *B*. Os métodos para escapar de labirintos — um conhecimento exigido dos ambiciosos aspirantes a heróis da antiguidade — também são exemplos de algoritmos de busca on-line. No entanto, a exploração espacial não é a única forma de exploração. Considere um bebê recém-nascido: ele tem muitas ações possíveis, mas não conhece os resultados de nenhuma delas e só experimentou alguns dos estados que tem possibilidade de alcançar. A descoberta gradual do bebê de como o mundo funciona é, em parte, um processo de busca on-line.

4.5.1 Problemas de busca on-line

Um problema de busca on-line só pode ser resolvido por um agente que executa ações, e não por computação pura. Assumiremos um ambiente determinístico e inteiramente observável (o Capítulo 17 faz o relaxamento dessas suposições), mas estipularemos que o agente sabe apenas o seguinte:

- AÇÕES(s), que devolve uma lista de ações permitidas no estado s.
- A função de custo do passo $c(s, a, s')$ — observe que isso não pode ser usado enquanto o agente não souber que s' é o resultado da ação e em s.
- TESTAR-OBJETIVO(s).

Em particular, observe que o agente *não pode* determinar o RESULTADO(s, a), exceto estando realmente em s e fazendo a. Por exemplo, no problema de labirinto mostrado na Figura 4.19, o agente não sabe que ir *Para cima* a partir de (1,1) leva a (1,2); nem sabe, tendo feito isso, que ir *Para baixo* o levará de volta a (1,1). Esse grau de ignorância pode ser reduzido em algumas aplicações — por exemplo, um robô explorador poderia saber como suas ações de movimentação funcionam e ser ignorante apenas sobre as posições dos obstáculos.

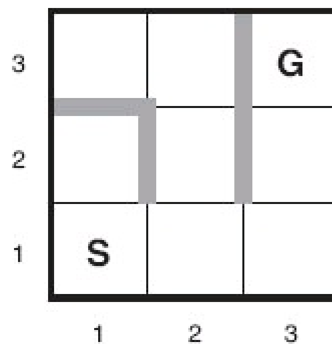


Figura 4.19 Um problema simples de labirinto. O agente inicia em S e deve chegar a G, mas nada sabe sobre o ambiente.

Finalmente, o agente poderia ter acesso a uma função heurística admissível $h(s)$ que avalia a distância desde o estado atual até um estado objetivo. Por exemplo, na Figura 4.19, o agente talvez conheça a posição do objetivo e seja capaz de usar a heurística da distância de Manhattan.

Em geral, o objetivo do agente é alcançar um estado objetivo ao mesmo tempo que minimiza o custo (outro objetivo possível é simplesmente explorar o ambiente inteiro). O custo é o custo total de caminho correspondente ao caminho que o agente de fato percorre. É comum comparar esse custo ao custo do caminho que o agente seguiria *se conhecesse o espaço de busca com antecedência*, isto é, o caminho real mais curto (ou a exploração completa mais curta). Na linguagem de algoritmos on-line, isso se denomina **razão competitiva**; que gostaríamos que fosse tão pequena quanto possível.

Embora isso soe como uma solicitação razoável, é fácil ver que a melhor razão competitiva que se pode alcançar é infinita em alguns casos. Por exemplo, se algumas ações forem **irreversíveis** — ou seja, conduzem a um estado do qual nenhuma ação leva de volta ao estado anterior —, a busca on-line poderá chegar acidentalmente a um estado de **beco sem saída**, a partir do qual nenhum estado objetivo será alcançável. Talvez você considere o termo "acidentalmente" pouco convincente — afinal, poderia existir um algoritmo que não tomasse o caminho do beco sem saída em sua exploração. Nossa afirmativa, para sermos mais precisos, é que *nenhum algoritmo pode evitar becos sem saída em todos os espaços de estados*. Considere os dois espaços de estados de becos

sem saída da Figura 4.20(a). Para um algoritmo de busca on-line que visitasse os estados S e A, os dois espaços de estados pareceriam *idênticos* e, assim, ele teria de tomar a mesma decisão em ambos. Por essa razão, ele falhará em um deles. Esse é um exemplo de **disputa adversarial** — podemos imaginar um oponente que constrói o espaço de estados, enquanto o agente o explora e que pode posicionar as metas e os becos sem saída onde desejar.

Figura 4.20 (a) Dois espaços de estados que poderiam levar um agente de busca on-line a um beco sem saída. Qualquer agente específico falhará em pelo menos um desses espaços. (b) Um ambiente bidimensional que pode fazer um agente de busca on-line seguir uma rota arbitrariamente ineficiente até o objetivo. Seja qual for a escolha do agente, o oponente bloqueará essa rota com outra parede longa e estreita, para que o caminho seguido seja muito mais longo que o melhor caminho possível.

Os becos sem saída constituem uma dificuldade real para a exploração de robôs — escadarias, rampas, precipícios e todos os tipos de terrenos naturais apresentam oportunidades para ações irreversíveis. Para progredir, simplesmente iremos supor que o espaço de estados é **explorável com segurança** — isto é, algum estado objetivo é alcançável a partir de todo estado alcançável. Os espaços de estados com ações reversíveis, como labirintos e quebra-cabeças de oito peças, podem ser vistos como grafos não orientados e sem dúvida são exploráveis com segurança.

Mesmo em ambientes exploráveis com segurança, nenhuma razão competitiva limitada poderá ser garantida se houver caminhos de custo ilimitado. É fácil mostrar isso em ambientes com ações irreversíveis, mas essa afirmativa também permanece verdadeira para o caso reversível, como mostra a Figura 4.20(b). Por essa razão, é comum descrever o desempenho de algoritmos de busca on-line em termos do tamanho do espaço de estados inteiro, e não apenas da profundidade do objetivo mais raso.

4.5.2 Agentes de busca on-line

Depois de cada ação, um agente on-line recebe uma percepção informando-o de qual estado ele alcançou; a partir dessa informação, ele pode ampliar seu mapa do ambiente. O mapa atual é usado

para decidir aonde ir em seguida. Essa intercalação de planejamento e ação significa que os algoritmos de busca on-line são bastante diferentes dos algoritmos de busca off-line que vimos anteriormente. Por exemplo, algoritmos off-line como A* podem expandir um nó em uma parte do espaço e depois expandir imediatamente um nó em outra parte do espaço porque a expansão de nós envolve ações simuladas, em vez de ações reais. Por outro lado, um algoritmo on-line pode descobrir sucessores para um nó que ele ocupa fisicamente. Para evitar percorrer todos os caminhos da árvore para expandir o próximo nó, parece melhor expandir nós em uma ordem *local*. A busca em profundidade tem exatamente essa propriedade porque (exceto quando ocorre retrocesso) o próximo nó expandido é um filho do nó expandido anterior.

Um agente de busca on-line em profundidade é mostrado na Figura 4.21. Esse agente armazena seu mapa em uma tabela, *Resultado*[*s*, *a*], que registra o estado resultante da execução da ação *a* no estado *s*. Sempre que uma ação para o estado atual não foi explorada, o agente experimenta essa ação. A dificuldade surge quando o agente tenta todas as ações em um estado. Na busca off-line em profundidade, o estado é simplesmente retirado da fila; em uma busca on-line, o agente tem de retroceder fisicamente. Na busca em profundidade, isso significa voltar para o estado a partir do qual o agente entrou no estado atual. Isso é conseguido mantendo-se uma tabela que lista, para cada estado, os estados predecessores aos quais o agente ainda não retrocedeu. Se o agente esgotar os estados aos quais ele pode retroceder, sua busca estará completa.

função AGENTE-DFS-ON-LINE(*s*′) **retorna** uma ação
  entradas: *s*′, uma percepção que identifica o estado atual
  persistente: *resultado*, uma tabela, indexada por ação e estado, inicialmente vazia
    experimentar, uma tabela que lista, para cada estado visitado, as ações ainda não tentadas
    retroceder, uma tabela que lista, para cada estado visitado, os retrocessos ainda não tentados
    s, *a*, o estado e a ação anteriores, inicialmente nulos
  se TESTE-OBJETIVO(*s*′) **então retornar** *parar*
  se *s* é um novo estado (não em *experimentar*) **então** *experimentar*[*s*′] ← AÇÕES(*s*′)
  se *s* é não nulo **então**
    resultado[*s*′, *a*] ← *s*′
    somar *s* ao início de *retrocesso*[*s*′]
  se *experimentar*[*s*′] é vazio **então**
    se *retrocesso*[*s*′] é vazio **então retornar** *parar*
    senão *a* ← uma ação *b* tal que *resultado*[*s*′, *b*] = DESEMPILHA(*retroceder*[*s*′])
  senão *a* ← DESEMPILHA(*experimentar*[*s*′])
  s ← *s*′
  retornar *a*

Figura 4.21 Um agente de busca on-line que utiliza exploração em profundidade. O agente só é aplicável em espaços de estados em que toda a ação pode ser “desfeita” por alguma outra ação.

Recomendamos que o leitor acompanhe o progresso do AGENTE-DFS-ON-LINE quando aplicado ao labirinto da Figura 4.19. É bastante fácil verificar que o agente acabará, no pior caso, percorrendo toda transição no espaço de estados exatamente duas vezes. Para a exploração, isso é ótimo; por outro lado, para encontrar um objetivo, a razão competitiva do agente poderia ser arbitrariamente ruim se resultasse em uma longa excursão quando houvesse um objetivo bem próximo ao estado inicial. Uma variante on-line do aprofundamento iterativo resolve esse problema; no caso de um ambiente que seja uma árvore uniforme, a razão competitiva de tal agente será uma constante pequena.

Em consequência de seu método de retrocesso, o AGENTE-DFS-ON-LINE só funcionará em espaços de estados nos quais as ações são reversíveis. Existem algoritmos um pouco mais complexos que funcionam em espaços de estados gerais, mas nenhum desses algoritmos tem uma razão competitiva limitada.

4.5.3 Busca local on-line

Assim como a busca em profundidade, a **busca de subida de encosta** tem a propriedade de localidade em suas expansões de nós. De fato, como ela mantém apenas um estado atual na memória, a busca de subida de encosta *já* é um algoritmo de busca on-line! Infelizmente, não é muito útil em sua forma mais simples porque deixa o agente parado em máximos locais, sem ter para onde ir. Além disso, os reinícios aleatórios não podem ser usados porque o agente não tem como se transportar para um novo estado.

Em vez de reinícios aleatórios, poderíamos considerar o uso de um **percurso aleatório** para explorar o ambiente. Um percurso aleatório simplesmente seleciona ao acaso uma das ações disponíveis do estado corrente; a preferência pode ser dada a ações que ainda não foram experimentadas. É fácil provar que um percurso aleatório irá *eventualmente* encontrar um objetivo ou completar sua exploração, desde que o espaço seja finito.[14] Por outro lado, o processo pode ser muito lento. A Figura 4.22 mostra um ambiente em que um percurso aleatório levará exponencialmente muitos passos para encontrar o objetivo porque, em cada passo, o progresso para trás é duas vezes mais provável que o progresso para frente. É claro que o exemplo é fictício, mas existem muitos espaços de estados reais cuja topologia resulta nesses tipos de "armadilhas" para percursos aleatórios.

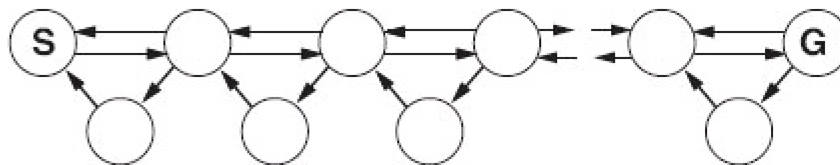


Figura 4.22 Um ambiente em que um percurso aleatório levará exponencialmente muitos passos para encontrar o objetivo.

A extensão da subida de encosta com *memória* em vez de aleatoriedade acaba sendo uma abordagem mais efetiva. A ideia básica é armazenar uma "melhor estimativa atual" $H(s)$ do custo para alcançar o objetivo a partir de cada estado que tenha sido visitado. $H(s)$ começa sendo apenas a

estimativa heurística $h(s)$ e é atualizada à medida que o agente ganha experiência no espaço de estados. A Figura 4.23 mostra um exemplo simples em um espaço de estados unidimensional. Em (a), o agente parece estar preso em um mínimo local plano no estado sombreado. Em vez de permanecer onde está, o agente deve seguir o que parece ser o melhor caminho até o objetivo, dadas as estimativas de custo atuais para seus vizinhos. O custo estimado para alcançar o objetivo através de um vizinho s' é o custo para chegar a s' somado ao custo estimado para ir de lá até um objetivo — isto é, $c(s, a, s') + H(s')$. No exemplo, existem duas ações com custos estimados 1 + 9 e 1 + 2, e assim parece melhor mover-se para a direita. Agora, é claro que a estimativa de custo 2 para o estado sombreado foi exageradamente otimista. Tendo em vista que o melhor movimento custa 1 e levou a um estado distante pelo menos dois passos de um objetivo, o estado sombreado deve estar pelo menos três passos distante de um objetivo e, portanto, seu H deve ser atualizado de acordo, como mostra a Figura 4.23(b). Continuando esse processo, o agente irá retroceder e avançar mais duas vezes, atualizando H em cada vez e "aplainando" o mínimo local até escapar para a direita.

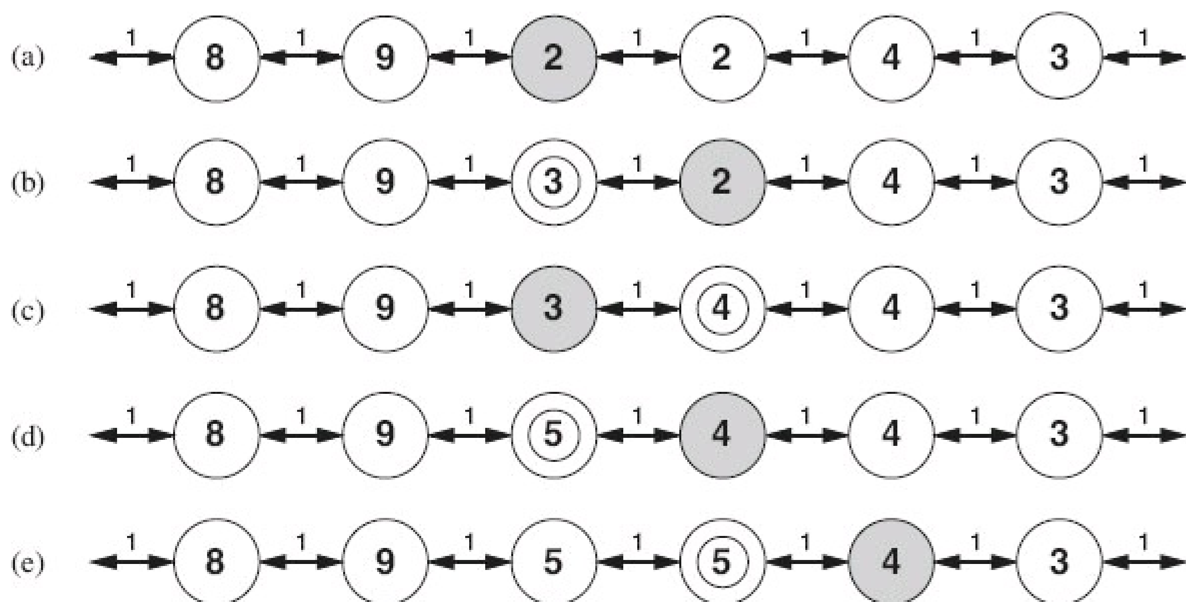


Figura 4.23 Cinco iterações de LRTA*em um espaço de estados unidimensional. Cada estado é identificado com $H(s)$, a estimativa de custo atual para alcançar um objetivo, e cada arco é identificado com seu custo do passo. O estado sombreado marca a posição do agente, e os valores atualizados estimados em cada iteração estão dentro de círculos.

Um agente que implementa esse esquema, chamado aprendizado em tempo real A* **LRTA*** (learning real-time A*), é mostrado na Figura 4.24. Da mesma forma que o AGENTE-BP-ON-LINE, ele constrói um mapa do ambiente usando a tabela *resultado*. Ele atualiza a estimativa de custo para o estado que acabou de deixar e depois escolhe o "aparentemente melhor" movimento de acordo com suas estimativas de custo atuais. Um detalhe importante é que sempre se supõe que ações ainda não tentadas em um estado s levam imediatamente ao objetivo com o menor custo possível, ou seja, $h(s)$. Esse **otimismo sob incerteza** encoraja o agente a explorar novos caminhos, possivelmente promissores.

função AGENTE-LRTA*(s') **retorna** uma ação
  entradas: s', uma percepção que identifica o estado corrente
  persistente: *resultado*, uma tabela, indexada por ação e estado, inicialmente vazia

```
                    H, uma tabela de estimativas de custo indexada pelo estado, inicialmente
                    vazia
                    s, a, o estado e a ação anteriores, inicialmente nulos
  se TESTE-OBJETIVO(s′) então retornar parar
  se s′ é um novo estado (não em H) então H[s′] ← h(s′)
  se s não é nulo
    resultado[s, a] ← s′
    H[s] ← min CUSTO-LRTA*(s, b, resultado[s, b], H)
                    b∈AÇÕES(s)
  a← uma ação b em AÇÕES(s′) que minimiza CUSTO-LRTA*(s′, b, resultado[s′, b], H)
  s ← s′
  retornar a
função CUSTO-LRTA*(s, a, s′, H) retorna uma estimativa de custo
  se s′ é indefinido então retornar h(s)
  senão retornar c(s, a, s′) + H[s′]
```

Figura 4.24 AGENTE-LRTA*seleciona uma ação de acordo com os valores de estados vizinhos, que são atualizados à medida que o agente se move no espaço de estados.

Um agente LRTA* oferece a garantia de encontrar um objetivo em qualquer ambiente finito explorável com segurança. Porém, diferentemente de A*, ele não é completo para espaços de estados infinitos — há casos em que ele pode ficar indefinidamente perdido. O agente pode explorar um ambiente de n estados em $O(n^2)$ passos no pior caso, mas com frequência funciona muito melhor. O agente LRTA* é apenas um em uma grande família de agentes on-line que podem ser definidos pela especificação da regra de seleção de ação e a regra de atualização de diferentes modos. Discutiremos essa família, desenvolvida originalmente para ambientes estocásticos, no Capítulo 21.

4.5.4 Aprendizado em busca on-line

A ignorância inicial dos agentes de busca on-line oferece várias oportunidades para aprendizado. Em primeiro lugar, os agentes aprendem um "mapa" do ambiente — mais precisamente, o resultado de cada ação em cada estado — apenas registrando cada uma de suas experiências (note que a suposição de ambientes determinísticos significa que uma experiência é suficiente para cada ação). Em segundo lugar, os agentes de busca local adquirem estimativas mais precisas do custo de cada estado usando regras de atualização local, como no LRTA*. No Capítulo 21, mostramos que essas atualizações convergem eventualmente para valores *exatos* em todo estado, desde que o agente explore o espaço de estados da maneira correta. Uma vez conhecidos os valores exatos podem ser tomadas decisões ótimas pela simples movimentação para o sucessor de valor mais baixo, isto é, a subida de encosta pura é então uma estratégia ótima.

Se você seguiu nossa sugestão para acompanhar o comportamento de AGENTE-DFS-ON-LINE no ambiente da Figura 4.19, terá notado que o agente não é muito brilhante. Por exemplo, depois de ver

que a ação *Para cima* vai de (1,1) para (1,2), o agente ainda não tem ideia de que a ação *Para baixo* volta a (1,1) ou de que a ação *Para cima* também vai de (2,1) para (2,2), de (2,2) para (2,3), e assim por diante. Em geral, gostaríamos que o agente aprendesse que *Para cima* aumenta a coordenada *y*, a menos que exista uma parede no caminho, que *Para baixo* a reduz, e assim por diante. Para que isso aconteça, primeiro precisamos de uma representação formal e explicitamente manipulável para esses tipos de regras gerais; até agora, ocultamos a informação contida na caixa-preta chamada função RESULTADO. A Parte III é dedicada a essa questão. Em segundo lugar, precisamos de algoritmos que possam construir regras gerais adequadas a partir das observações específicas feitas pelo agente. Esses assuntos serão estudados no Capítulo 18.

4.6 RESUMO

Este capítulo examinou algoritmos de busca para problemas além do caso "clássico" de encontrar o caminho mais curto para um objetivo em um ambiente observável, determinístico, discreto.

- Métodos de *busca local* como **subida de encosta** operam sobre formulações de estados completos, mantendo na memória apenas um pequeno número de nós. Foram desenvolvidos vários algoritmos estocásticos, incluindo a **têmpera simulada**, que devolve soluções ótimas quando recebe um cronograma de resfriamento apropriado.
- Muitos métodos de busca local também se aplicam a problemas de espaços contínuos. A **programação linear** e a **otimização convexa** obedecem a certas restrições sobre a forma do espaço de estados e da natureza da função objetivo, e admitem algoritmos de tempo polinomial que são sempre extremamente eficientes na prática.
- Um **algoritmo genético** é uma busca de subida de encosta estocástica em que é mantida uma grande população de estados. Novos estados são gerados por **mutação** e por **cruzamento**, que combinam pares de estados da população.
- Em ambientes não determinísticos, os **agentes** podem aplicar pesquisa E-OU para gerar planos de **contingência** que alcançam o objetivo, independentemente do resultado que ocorre durante a execução.
- Quando o ambiente for parcialmente observável, o **estado de crença** representa o conjunto de estados possíveis em que o agente pode estar.
- Os algoritmos de busca-padrão podem ser aplicados diretamente ao espaço de estado de crença para resolver **problemas sem sensoriamento**, e os de busca de estado de crença E-OU podem resolver problemas gerais parcialmente observáveis. Os algoritmos incrementais que constroem soluções estado por estado em um estado de crença são muitas vezes mais eficientes.
- Os **problemas de exploração** surgem quando o agente não tem nenhuma ideia sobre os estados e ações de seu ambiente. No caso de ambientes exploráveis com segurança, agentes de **busca on-line** podem construir um mapa e encontrar um objetivo, se existir algum. A atualização de estimativas heurísticas a partir da experiência fornece um método efetivo para escapar de mínimos locais.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

As técnicas de busca local têm uma longa história em matemática e ciência da computação. Na realidade, o método de Newton-Raphson (Newton, 1671; Raphson, 1690) pode ser visto como um método de busca local muito eficiente para espaços contínuos em que as informações de gradiente estão disponíveis. Brent (1973) é uma referência clássica para algoritmos de otimização que não exigem tais informações. A busca em feixe, que apresentamos como algoritmo de busca local, teve origem como uma variante de largura limitada da programação dinâmica para reconhecimento de voz no sistema HARPY (Lowerre, 1976). Um algoritmo relacionado é analisado em profundidade por Pearl (1984, Capítulo 5).

O tópico de busca local foi revigorado no início dos anos 1990 por resultados surpreendentemente bons para problemas de satisfação de restrições como o das *n* rainhas (Minton *et al*., 1992) e raciocínio lógico (Selman *et al*., 1992), e pela incorporação da aleatoriedade, de múltiplas buscas simultâneas e de outros aperfeiçoamentos. Esse renascimento do que Christos Papadimitriou chamou de algoritmos da "Nova Era" também despertou interesse crescente entre os cientistas da computação teórica (Koutsoupias e Papadimitriou, 1992; Aldous e Vazirani, 1994). No campo da pesquisa operacional, uma variante da subida de encosta chamada **busca tabu** ganhou popularidade (Glover e Laguna, 1997). Esse algoritmo mantém uma lista tabu de *k* estados visitados anteriormente que não podem ser revisitados; essa lista tanto pode melhorar a eficiência na busca em grafos, como pode permitir que o algoritmo escape de alguns mínimos locais. Outra melhoria útil em relação à subida de encosta é o algoritmo STAGE (Boyan e Moore, 1998). A ideia é usar os máximos locais encontrados pela subida de encosta com reinício aleatório para ter uma ideia da forma geral da topologia. O algoritmo ajusta uma superfície suave ao conjunto de máximos locais e depois calcula analiticamente o máximo global dessa superfície. Este se torna o novo ponto de reinício. Demonstrou-se que o algoritmo funciona na prática em problemas difíceis. Gomes *et al*. (1998) mostraram que as distribuições de tempo de execução de algoritmos de retrocesso sistemático com frequência têm **distribuição de cauda pesada**; isso significa que a probabilidade de um tempo de execução muito longo é maior do que seria previsto se os tempos de execução estivessem exponencialmente distribuídos. Quando a distribuição do tempo de execução é de cauda pesada, o reinício aleatório encontra em média uma solução mais rápida do que uma única execução para a conclusão.

A têmpera simulada foi descrita primeiro por Kirkpatrick *et al*. (1983), que a tomaram emprestada diretamente do **algoritmo de Metropolis** (usado para simular sistemas complexos em física — Metropolis *et al*., 1953 — e foi criada supostamente durante um jantar festivo em Los Alamos). A têmpera simulada agora é um campo em si mesmo, com centenas de artigos publicados a cada ano.

Encontrar soluções ótimas em espaços contínuos é o principal assunto de diversos campos, incluindo a **teoria de otimização**, a **teoria de controle ótimo** e o **cálculo de variações**. As técnicas básicas são bem explicadas por Bishop (1995); Press *et al*. (2007) cobrem uma vasta gama de algoritmos e fornecem o software correspondente.

Como Andrew Moore indicou, os pesquisadores tiveram inspiração pelos algoritmos de busca e otimização de uma ampla variedade de áreas de estudo: metalurgia (têmpora simulada), biologia (algoritmos genéticos), economia (algoritmos baseados no mercado de ações), entomologia

(otimização de colônia de formigas), neurologia (redes neurais), comportamento animal (aprendizagem por reforço), montanhismo (subida de encosta) e outros.

A **programação linear** (PL) foi inicialmente estudada sistematicamente pelo matemático russo Leonid Kantorovich (1939). Foi uma das primeiras aplicações de computadores; o **algoritmo simplex** (Dantzig, 1949) ainda é usado, apesar da complexidade exponencial do pior caso. Karmarkar (1984) desenvolveu a família muito mais eficiente de métodos de **ponto interior**, que demonstrou ter complexidade polinomial para a classe mais geral de problemas de otimização convexa por Nesterov e Nemirovski (1994). São fornecidas excelentes introduções à otimização convexa por Ben-Tal e Nemirovski (2001) e Boyd e Vandenberghe (2004).

O trabalho de Sewall Wright (1931) sobre o conceito de uma **topologia de adaptação** foi um importante precursor para o desenvolvimento de algoritmos genéticos. Na década de 1950, diversos estatísticos, incluindo Box (1957) e Friedman (1959), utilizaram técnicas evolucionárias em problemas de otimização, mas somente quando Rechenberg (1965) introduziu as **estratégias de evolução** para resolver problemas de otimização de aerofólios a abordagem ganhou popularidade. Nas décadas de 1960 e 1970, John Holland (1975) defendeu os algoritmos genéticos, não só como uma ferramenta útil, mas também como um método para expandir nossa compreensão da adaptação, biológica ou não (Holland, 1995). O movimento de **vida artificial** (Langton, 1995) leva essa ideia um passo adiante, visualizando os produtos de algoritmos genéticos como *organismos*, em vez de soluções para problemas. O trabalho nesse campo desenvolvido por Hinton e Nowlan (1987) e por Ackley e Littman (1991) foi realizado principalmente para esclarecer as implicações do efeito de Baldwin. Para um conhecimento geral sobre os fundamentos da evolução, recomendamos Smith e Szathmáry (1999), Ridley (2004) e Carroll (2007).

A maioria das comparações de algoritmos genéticos com outras abordagens (em especial a subida de encosta estocástica) revelou que os algoritmos genéticos convergem mais lentamente (O'Reilly e Oppacher, 1994; Mitchell *et al*., 1996; Juels e Wattenberg, 1996; Baluja, 1997). Tais descobertas não são universalmente populares dentro da comunidade de AG, mas tentativas recentes dentro dessa comunidade para entender a busca baseada na população como uma forma aproximada de aprendizado bayesiano (veja o Capítulo 20) talvez ajudem a reduzir o abismo entre o campo e suas críticas (Pelikan *et al*., 1999). A teoria de **sistemas quadráticos dinâmicos** também pode explicar o desempenho dos AGs (Rabani *et al*., 1998). Veja em Lohn *et al*. (2001) um exemplo de AG aplicado ao projeto de antenas e, em Renner e Ekart (2003), para uma aplicação de projeto assistido por computador.

O campo de **programação genética** está intimamente relacionado aos algoritmos genéticos. A principal diferença é que as representações que sofrem mutações e combinações são programas, em vez de cadeias de bits. Os programas são representados sob a forma de árvores de expressões; as expressões podem estar em uma linguagem padrão como Lisp ou podem ser projetadas especificamente para representar circuitos, controladores de robôs, e assim por diante. O cruzamento envolve a união de subárvores, e não de subcadeias. Essa forma de mutação garante que os descendentes serão expressões bem formadas, o que não ocorreria se os programas fossem manipulados como cadeias.

O recente interesse em programação genética foi incentivado pelo trabalho de John Koza (Koza, 1992, 1994), mas remonta pelo menos aos primeiros experimentos com código de máquina

realizados por Friedberg (1958) e com autômatos de estados finitos, desenvolvidos por Fogel *et al*. (1966). Como no caso de algoritmos genéticos, existe um debate sobre a eficácia da técnica. Koza *et al*. (1999) descrevem experimentos no projeto automatizado dos dispositivos de circuitos utilizando programação genética.

Os periódicos *Evolutionary Computation* e *IEEE Transactions on Evolutionary Computation* estudam algoritmos genéticos e programação genética; também são encontrados artigos em *Complex Systems*, *Adaptive Behavior* e *Artificial Life*. A conferência principal é a *Genetic and Evolutionary Computation Conference* (GECCO). Mitchell (1996), Fogel (2000) e Langdon e Poli (2002), e o livro on-line gratuito de Poli *et al.* (2008), oferecem bons textos de visão geral sobre algoritmos genéticos.

A imprevisibilidade e a observabilidade parcial de ambientes reais foram reconhecidas no início de projetos de robótica que utilizavam técnicas de planejamento, incluindo Shakey (Fikes *et al*., 1972) e Freddy (Michie, 1974). Os problemas receberam mais atenção após a publicação do artigo influente de McDermott (1978a), *Planning e Acting*.

O primeiro trabalho a fazer uso explícito de árvores E-OU parece ter sido o programa SAINT de Slagle para a integração simbólica, mencionado no Capítulo 1. Amarel (1967) aplicou a ideia de prova de teorema proposicional, um tópico que será discutido no Capítulo 7, e introduziu um algoritmo de busca semelhante à BUSCA-EM-GRAFOS-E-OU. O algoritmo foi desenvolvido mais adiante e formalizado por Nilsson (1971), que também descreveu AO*, que, como seu nome sugere, encontra soluções ótimas dada uma heurística admissível. AO* foi analisado e melhorado por Martelli e Montanari (1973). AO* é um algoritmo top-down; uma generalização bottom-up de A* é A*LD, isto é, A* Lightest Derivation (Felzenszwalb e McAllester, 2007). O interesse pela busca E-OU passou por um renascimento nos últimos anos, com novos algoritmos para encontrar soluções cíclicas (Jimenez e Torras, 2000; Hansen e Zilberstein, 2001) e novas técnicas inspiradas por programação dinâmica (Bonet e Geffner, 2005).

A ideia de transformar problemas parcialmente observáveis em problemas de estado de crença originou com Astrom (1965) para o caso muito mais complexo de incerteza probabilística (veja o Capítulo 17). Erdmann e Mason (1988) estudaram o problema da manipulação robótica sem sensores, usando uma forma contínua de busca de estado de crença. Eles mostraram que era possível orientar uma peça em uma mesa a partir de uma posição inicial arbitrária por uma sequência bem concebida de ações pendulares. Métodos mais práticos, com base em uma série de barreiras diagonais precisamente orientadas através de uma correia transportadora, utilizam o mesmo critério algorítmico (Wiegley *et al*., 1996).

A abordagem do estado de crença foi reinventada no contexto de problemas de busca sem sensoriamento e parcialmente observáveis por Genesereth e Nourbakhsh (1993). Foi realizado um trabalho adicional sobre os problemas sem sensoriamento na comunidade de planejamento baseado em lógica (Goldman e Boddy, 1996; Smith e Weld, 1998). Esse trabalho enfatizou representações concisas para os estados de crença, como explicado no Capítulo 11. Bonet e Geffner (2000) introduziram as primeiras heurísticas eficazes para a busca do estado de crença, que foram refinados por Bryce *et al*. (2006). A abordagem incremental da busca do estado de crença, em que as soluções são construídas de forma incremental para subconjuntos de estados dentro de cada estado de crença, foi estudada na literatura de planejamento por Kurien *et al*. (2002); vários novos algoritmos

incrementais foram apresentados para problemas não determinísticos, parcialmente observáveis por Russell e Wolfe (2005). Referências adicionais para planejamento, em ambientes estocásticos parcialmente observáveis, aparecem no Capítulo 17.

Os algoritmos para explorar espaços de estados desconhecidos têm despertado interesse por muitos séculos. A busca em profundidade em um labirinto pode ser implementada mantendo-se a mão esquerda na parede; os ciclos podem ser evitados marcando-se cada junção. A busca em profundidade falha com ações irreversíveis; o problema mais geral de exploração de **grafos eulerianos** (isto é, grafos em que cada nó tem números iguais de arestas de entrada e saída) foi resolvido por um algoritmo criado por Hierholzer (1873). O primeiro estudo algorítmico completo do problema de exploração de grafos arbitrários foi proposto por Deng e Papadimitriou (1990), que desenvolveram um algoritmo completamente geral, mas mostraram que não é possível nenhuma razão competitiva limitada para explorar um grafo geral. Papadimitriou e Yannakakis (1991) examinaram a questão de encontrar caminhos até um objetivo em ambientes de planejamento de caminhos geométricos (em que todas as ações são reversíveis). Eles mostraram que uma pequena razão competitiva pode ser alcançada com obstáculos quadrados, mas que não é possível alcançar nenhuma razão limitada com obstáculos retangulares em geral (veja a Figura 4.20).

O algoritmo LRTA* foi desenvolvido por Korf (1990) como parte de uma investigação da **busca em tempo real** para ambientes em que o agente deve atuar depois de buscar apenas durante um período de tempo fixo (uma situação muito mais comum em jogos com dois participantes). O LRTA* é de fato um caso especial de algoritmos de aprendizado de reforço para ambientes estocásticos (Barto *et al*., 1995). Sua política de otimismo sob incerteza — sempre se dirigir para o estado não visitado mais próximo — pode resultar em um padrão de exploração menos eficiente no caso não informado do que a simples busca em profundidade (Koenig, 2000).

Dasgupta *et al*. (1994) mostram que a busca de aprofundamento iterativo on-line é otimamente eficiente para encontrar um objetivo em uma árvore uniforme sem informações heurísticas. Diversas variantes informadas sobre o tema do LRTA* foram desenvolvidas com diferentes métodos de busca e atualização dentro da parte conhecida do grafo (Pemberton e Korf, 1992). Até agora, não existe uma boa compreensão de como encontrar objetivos com eficiência ótima quando se utilizam informações heurísticas.

EXERCÍCIOS

4.1 Forneça o nome do algoritmo que resulta de cada um dos seguintes casos especiais:

a. Busca em feixe local com $k = 1$.

b. Busca em feixe local com um estado inicial e nenhum limite sobre o número de estados mantidos.

c. Tempêra simulada com T = 0 em todas passos (com omissão do teste de término).

d. Têmpora simulada com $T = \infty$ em todos os passos.

e. Algoritmo genético com tamanho de população $N = 1$.

4.2 O Exercício 3.16 considera o problema da construção de ferrovias sob a suposição de que as peças se encaixam exatamente sem nenhuma folga. Agora considere o problema real, em que as peças não se encaixam exatamente, mas permitem até 10 graus de rotação para ambos os lados do alinhamento "apropriado". Explique como formular o problema para que ele possa ser resolvido por têmpora simulada.

4.3 Neste exercício, exploramos a utilização de métodos de busca local para resolver TSP's do tipo definido no Exercício 3.30.

a. Implemente e teste um método de subida de encosta para resolver TSP's. Compare os resultados com soluções ótimas obtidas do algoritmo A* com a heurística da MST (Minimum-Spanning-Tree) (Exercício 3.30).

b. Repita parte (a) utilizando um algoritmo genético em vez de um de subida de encosta. Consulte Larrañaga et al. (1999) se desejar alguma sugestão de representações.

4.4 Gere um grande número de instâncias do quebra-cabeça de oito peças e de oito rainhas e resolva-as (quando possível) por subida de encosta (com variantes de subida mais íngreme e primeira escolha), por subida de encosta com reinício aleatório e por têmpera simulada. Meça o custo da busca e a porcentagem de problemas resolvidos e elabore um gráfico desses valores contra o custo da solução ótima. Comente seus resultados.

4.5 O algoritmo de BUSCA-EM-GRAFOS-E-OU na Figura 4.11 verifica os estados repetidos apenas no caminho da raiz até o estado atual. Suponha que, além disso, o algoritmo armazene *cada* estado visitado numa lista (veja a BUSCA-EM-LARGURA na Figura 3.11 como exemplo). Determine a informação que deveria ser armazenada e como o algoritmo deveria utilizar essa informação quando um estado repetido fosse encontrado. (*Dica*: Será necessário pelo menos distinguir estados para os quais um subplano bem-sucedido foi anteriormente construído e os estados para os quais nenhum subplano pode ser encontrado.) Explique como utilizar rótulos, conforme definido na Seção 4.3.3, para evitar ter várias cópias de subplanos.

4.6 Explique exatamente como modificar o algoritmo BUSCA-EM-GRAFOS-E-OU para gerar um plano cíclico se não existir nenhum plano acíclico. Será necessário lidar com três questões: rotulagem das etapas do plano para que um plano cíclico possa apontar para uma parte anterior do plano, modificando a BUSCA-OU para que continue a procurar por planos acíclicos depois de encontrar um plano cíclico e aumentando a representação do plano para indicar se um plano é cíclico. Mostre como o algoritmo funciona (a) no mundo do aspirador de pó com incerteza e (b) no mundo do aspirador de pó defeituoso e incerto. Faça a implementação para verificar os resultados.

4.7 Na Seção 4.4.1 apresentamos os estados de crença para resolver problemas de busca sem sensoriamento. Uma sequência de ações resolve um problema sem sensoriamento se ele mapear cada estado físico no estado de crença inicial b para um estado objetivo. Suponha que o agente conheça $h^*(s)$, o custo verdadeiro ideal para resolver o estado físico s em um problema completamente observável, para cada estado s em b. Encontre uma heurística admissível $h(b)$ para o problema sem sensoriamento em termos desses custos e prove a sua admissibilidade. Comente sobre a precisão dessa heurística para o problema do aspirador de pó sem sensoriamento da Figura 4.14. Quão bom é

o desempenho do A*?

4.8 Este exercício explora as relações subconjunto-superconjunto entre os estados de crença em ambientes sem sensoriamento ou parcialmente observáveis.

a. Prove que, se uma sequência de ações for uma solução para um estado de crença *b*, é também uma solução para qualquer subconjunto de *b*. Pode-se dizer algo sobre os superconjuntos de *b*?

b. Explique em detalhe como modificar a busca em grafos para problemas sem sensoriamento para tirar proveito da resposta em (a).

c. Explique em detalhes como modificar a busca E-OU para problemas parcialmente observáveis, além das modificações descritas em (b).

4.9 Nas Seção 4.4.1 foi pressuposto que determinada ação teria o mesmo custo quando executada em qualquer estado físico dentro de determinado estado de crença (isso leva a um problema de busca em estado de crença com custos de passos bem definidos). Agora, considere o que acontece quando essa suposição é feita. A noção de otimalidade ainda faz sentido nesse contexto ou requer modificação? Considere também várias definições possíveis do "custo" de executar uma ação em um estado de crença; por exemplo, poderíamos usar o *mínimo* de custos físicos ou o *máximo*, ou um *intervalo* de custo com o limite inferior sendo o custo mínimo e o limite superior o máximo, ou apenas manter o conjunto de todos os custos possíveis para essa ação. Para cada um deles, explore se A* (com alterações, se necessário) pode devolver soluções ótimas.

4.10 Considere a versão sem sensoriamento do mundo do aspirador de pó defeituoso. Desenhe o espaço de estado de crença acessível a partir do estado de crença inicial {1, 3, 5, 7} e explique por que o problema é insolúvel.

4.11 Podemos transformar o problema de navegação do Exercício 3.7 em um ambiente como o seguinte:

- A percepção será uma lista de posições, *em relação ao agente*, dos vértices visíveis. A percepção *não* inclui a posição do robô! O robô deve aprender a sua própria posição a partir do mapa; por ora, assuma que cada localidade tem uma "visão" diferente.
- Cada ação será um vetor que descreve um caminho em linha reta a ser seguido. Se o caminho estiver desobstruído, a ação será bem-sucedida; caso contrário, o robô vai parar no primeiro ponto em que seu caminho cruzar com um obstáculo. Se o agente devolve um vetor de movimento zero e estiver no objetivo (que é fixo e conhecido), o ambiente teletransporta o agente para uma *localidade aleatória* (não dentro de um obstáculo).
- A medida de desempenho penaliza o agente de um ponto para cada unidade de distância percorrida e premia em 1.000 pontos cada vez que o objetivo for alcançado.

a. Implemente esse ambiente e um agente de resolução de problema para ele. Após cada teletransporte, o agente terá que formular um novo problema, que envolve descobrir a sua localização atual.

b. Documente o desempenho do seu agente (faça o agente gerar comentários apropriados à medida que se move ao redor) e relate o seu desempenho ao longo de 100 episódios.

c. Modifique o ambiente de modo que 30% das vezes o agente acabe em um destino não

pretendido (escolhido aleatoriamente a partir de outros vértices visíveis, se houver algum; caso contrário, não há movimento algum). Esse é um modelo grosseiro de erros de movimento de um robô real. Modifique o agente para que, quando tal erro for detectado, ele descubra onde está e, em seguida, construa um plano de voltar para onde estava e retome ao plano antigo. Lembre-se de que, por vezes, voltar para onde estava também pode falhar! Mostre um exemplo do agente superando com sucesso dois erros sucessivos de movimento e ainda alcançando seu objetivo.

d. Agora tente dois esquemas diferentes de recuperação após um erro: (1) dirija-se para o vértice mais próximo da rota original; e (2) replaneje uma rota para o objetivo a partir da nova localização. Compare o desempenho dos três esquemas de recuperação. A inclusão de custos de busca afetaria a comparação?

e. Agora, suponha que existam localizações onde a visão seja idêntica (por exemplo, suponha que o mundo seja uma grade com obstáculos quadrados). Que tipo de problema o agente enfrentaria agora? Com o que as soluções se parecem?

4.12 Suponha que um agente esteja em um ambiente de labirinto 3 × 3 como o da Figura 4.19. O agente sabe que sua posição inicial é (1,1), que o objetivo está em (3,3) e que as quatro ações *Para cima*, *Para baixo*, *Esquerda*, *Direita* têm seus efeitos habituais, a menos que sejam bloqueadas por uma parede. O agente *não* sabe onde estão as paredes internas. Em qualquer estado específico, o agente percebe o conjunto de ações válidas; ele também pode saber se o estado já foi visitado antes ou é um novo estado.

a. Explique como esse problema de busca on-line pode ser visualizado como uma busca off-line no espaço de estados de crença, onde o estado de crença inicial inclui todas as configurações possíveis de ambiente. Qual é o tamanho do estado de crença inicial? Qual é o tamanho do espaço de estados de crença?

b. Quantas percepções distintas são possíveis no estado inicial?

c. Descreva as primeiras ramificações de um plano de contingência para esse problema. Qual é o tamanho (aproximado) do plano completo?

Note que esse plano de contingência é uma solução para *todo ambiente possível* que se ajusta à descrição dada. Portanto, a intercalação de busca e execução não é estritamente necessária, nem mesmo em ambientes desconhecidos.

4.13 Neste exercício, examinaremos a subida de encosta no contexto da navegação de robôs, usando o ambiente da Figura 3.31 como exemplo.

a. Repita o Exercício 4.11 usando subida de encosta. Seu agente ficará preso em um mínimo local? É *possível* que ele fique preso com obstáculos convexos?

b. Construa um ambiente poligonal não convexo no qual o agente fique preso.

c. Modifique o algoritmo de subida de encosta de forma que, em vez de realizar uma busca de profundidade 1 a fim de decidir para onde ir em seguida, ele realize uma busca de profundidade *k*. Ele deve encontrar o melhor caminho de *k* passos, percorrer um passo ao longo dele e depois repetir o processo.

d. Existe algum *k* para o qual o novo algoritmo oferece a garantia de escapar de mínimos locais?

e. Explique como o RTDA* permite ao agente escapar de mínimos locais nesse caso.

4.14 Como DFS, o DFS on-line é incompleto em espaços de estado reversíveis com caminhos infinitos. Por exemplo, suponha que os estados sejam pontos em uma grade infinita bidimensional e as ações sejam vetores unitários (1,0), (0,1), (-1,0), (0,-1), nessa ordem. Mostre que a DFS on-line iniciando em (0,0) não irá alcançar (1,-1). Em adição a esse estado atual, suponha que o agente possa observar todos os estados sucessores e as ações que irão conduzir a eles. Escreva um algoritmo que seja completo mesmo para espaços de estado bidirecionais com caminhos infinitos.Que estados ele visita ao alcançar (1, -1)?

---

[1] Gerar um estado *aleatório* a partir de um espaço de estados implicitamente especificado pode ser um problema difícil por si só.

[2] Luby *et al.* (1993) provam que é melhor, em alguns casos, reiniciar um algoritmo de busca aleatória depois de um período de tempo fixo e específico, e isso pode ser *muito* mais eficiente que deixar cada busca continuar indefinidamente. Proibir ou limitar o número de movimentos laterais é um exemplo dessa ideia.

[3] A busca em feixe local é uma adaptação da **busca em feixe** (**beam search**), que é um algoritmo baseado em caminhos.

[4] Existem muitas variantes dessa regra de seleção. Pode-se mostrar que o método de **culling** (corte, eliminação), no qual todos os indivíduos abaixo de determinado limiar são descartados, converge com maior rapidez que a versão aleatória (Baum *et al.*, 1995).

[5] É nessa situação que a codificação é importante. Se for usada uma codificação de 24 bits em vez de oito dígitos, o ponto de cruzamento terá uma chance de 2/3 de estar no meio de um dígito, o que resulta em uma mutação essencialmente arbitrária desse dígito.

[6] Um conhecimento básico de cálculo multivariado e aritmética vetorial será útil durante a leitura desta seção.

[7] Em geral, a atualização Newton-Raphson pode ser vista como um ajuste a uma superfície quadrática para f em $\mathbf{x}$ e então se movendo diretamente para o mínimo daquela superfície, que é também o mínimo de f se f for quadrático.

[8] Um conjunto de pontos S é convexo se a linha que une dois pontos quaisquer em S também estiver contido em S. Uma **função convexa** é aquele da qual o espaço "acima" forma um conjunto convexo; por definição, funções convexas não têm um mínimo local (em oposição ao global).

[9] Supomos que a maioria dos leitores, por enfrentar problemas semelhantes, se identifica com o nosso agente. Pedimos desculpas aos proprietários de utensílios domésticos modernos, eficientes, que não podem tirar vantagem desse exemplo pedagógico.

[10] Em um ambiente totalmente observável, cada estado de crença contém um estado físico. Assim, podemos ver os algoritmos do Capítulo 3 como se estivessem em busca de um espaço de estado de crença de estados de crença singulares.

[11] Aqui, e ao longo do livro, o "acento circunflexo" em $\hat{b}$ significa um valor estimado ou previsto para b.

[12] Pedimos desculpas para aqueles que não estão familiarizados com o efeito de crianças pequenas num ambiente.

[13] O termo "on-line" é de uso comum em ciência da computação para fazer referência a algoritmos que devem processar dados de entrada à medida que eles são recebidos, em vez de esperar que o conjunto de dados de entrada inteiro se torne disponível.

[14] Os percursos aleatórios são completos em grades infinitas unidimensionais e bidimensionais. Em uma grade tridimensional, a probabilidade de alguma vez o percurso retornar ao ponto de partida é apenas de cerca de 0,3405 (Hughes, 1995).

CAPÍTULO 5

Busca competitiva

Em que examinamos os problemas que surgem quando tentamos planejar com antecedência em um mundo no qual outros agentes estão fazendo planos contra nós.

5.1 JOGOS

O Capítulo 2 examinou os **ambientes multiagentes**, em que cada agente precisa considerar as ações de outros agentes e o modo como essas ações afetam seu próprio bem-estar. A imprevisibilidade desses outros agentes pode introduzir muitas **contingências** possíveis no processo de resolução de problemas do agente, conforme discutido no Capítulo 4. Neste capítulo, abordaremos ambientes **competitivos**, em que os objetivos dos agentes estão em conflito, dando origem a problemas de **busca competitiva** — frequentemente conhecidos como **jogos**.

A **teoria de jogos** (matemática), um ramo da economia, visualiza qualquer ambiente multiagente como um jogo, desde que o impacto de cada agente sobre os outros seja "significativo", não importando se os agentes são cooperativos ou competitivos.[1] Em IA, os "jogos" mais comuns são de um tipo bastante especializado — que os teóricos de jogos denominam jogos determinísticos de revezamento de dois jogadores **de soma zero** com **informações perfeitas** (como o xadrez). Em nossa terminologia, isso significa ambientes determinísticos completamente observáveis em que dois agentes agem alternadamente e em que os valores de utilidade no fim do jogo são sempre iguais e opostos (ou simétricos). Por exemplo, se um jogador ganha um jogo de xadrez, o outro jogador necessariamente perde. É essa oposição entre as funções utilidade dos agentes que gera a situação de competição.

Os jogos ocuparam as faculdades intelectuais dos seres humanos — chegando algumas vezes a um grau alarmante — desde que surgiu a civilização. Para pesquisadores de IA, a natureza abstrata dos jogos os torna um assunto atraente para estudo. É fácil representar o estado de um jogo e, em geral, os agentes se restringem a um pequeno número de ações cujos resultados são definidos por regras precisas. Jogos físicos, como críquete e hóquei sobre o gelo, têm descrições muito mais complicadas, uma faixa muito maior de ações possíveis e regras bastante imprecisas definindo a legalidade das ações. Com exceção do futebol de robôs, esses jogos físicos não atraíram muito interesse na comunidade de IA.

Os jogos, diferentemente da maior parte dos miniproblemas estudados no Capítulo 3, são interessantes *porque* são muito difíceis de resolver. Por exemplo, o xadrez tem um fator médio de ramificação de cerca de 35, e as partidas com frequência chegam até a 50 movimentos por cada jogador; assim, a árvore de busca tem aproximadamente 35^{100} ou 10^{154} nós (embora o grafo de busca tenha "apenas" cerca de 10^{40} nós distintos). Os jogos, como o mundo real, exigem portanto a habilidade de tomar *alguma* decisão, até mesmo quando o cálculo da decisão *ótima* é inviável. Os jogos também penalizam a ineficiência de forma severa. Enquanto uma implementação de busca A* com a metade da eficiência levará duas vezes mais para executar até a conclusão, um programa de xadrez com metade da eficiência no uso de seu tempo disponível provavelmente será derrubado sem piedade, mantendo-se outros aspectos inalterados. Por essa razão, a busca de jogos elaborou várias ideias interessantes sobre como fazer o melhor uso possível do tempo.

Começamos com uma definição do movimento ótimo e um algoritmo para descobri-lo. Em seguida, examinaremos técnicas para escolher um bom movimento quando o tempo é limitado. A **poda** nos permite ignorar partes da árvore de busca que não fazem diferença para a escolha final, e as **funções de avaliação** de heurísticas nos oferecem a oportunidade de fazer uma aproximação da verdadeira utilidade de um estado sem realizar uma busca completa. A Seção 5.5 discute jogos como gamão, que incluem um elemento de sorte; também discutimos o jogo de *bridge*, que inclui elementos de **informação imperfeita** porque nem todas as cartas estão visíveis para cada jogador. Por fim, veremos como os programas de jogos de última geração se comportam contra a oposição humana e, ainda, orientações para desenvolvimentos futuros.

Primeiro consideraremos jogos com dois jogadores, que chamaremos MAX e MIN por motivos que logo ficarão óbvios. MAX faz o primeiro movimento, e depois eles se revezam até o jogo terminar. No fim do jogo, os pontos são dados ao jogador vencedor e são impostas penalidades ao perdedor. Um jogo pode ser definido formalmente como uma espécie de problema de busca com os seguintes componentes:

- S_0: o **estado inicial**, que especifica como o jogo é criado no início.
- JOGADORES(s): define qual jogador deve se mover em um estado.
- AÇÕES(s): retornam o conjunto de movimentos válidos em um estado.
- RESULTADO(s, a): o **modelo de transição** que define o resultado de um movimento.
- TESTE DE TÉRMINO(s): um **teste de término**, que é verdadeiro quando o jogo termina e, do contrário, falso. Os estados em que o jogo é encerrado são chamados **estados terminais**.
- UTILIDADE(s, p): uma **função utilidade** (também chamada função objetivo ou função compensação) define o valor numérico para um jogo que termina no estado terminal s por um jogador p. No xadrez, o resultado é uma vitória, uma derrota ou um empate, com valores +1, 0 ou $\frac{1}{2}$. Alguns jogos têm uma variedade mais ampla de resultados possíveis; a compensação no gamão varia de 0 até +192. Um **jogo de soma zero** é (confusamente) definido como aquele em que a compensação total para todos os jogadores é a mesma para cada instância do jogo. O xadrez é de soma zero porque cada jogo tem compensação $0 + 1$, $1 + 0$ ou $\frac{1}{2} + \frac{1}{2}$. "Soma constante" teria sido um termo melhor, mas soma zero é tradicional e faz sentido se você imaginar que de cada jogador é cobrada uma taxa de entrada de $\frac{1}{2}$.

O estado inicial função AÇÕES e a função RESULTADO definem a **árvore de jogo** correspondente ao jogo — uma árvore onde os nós são estados do jogo e as bordas são movimentos. A Figura 5.1 mostra parte da árvore de jogo para o jogo da velha. A partir do estado inicial, MAX tem nove movimentos possíveis. O jogo se alterna entre a colocação de um X por MAX e a colocação de um O por MIN até alcançarmos nós de folhas correspondentes a estados terminais, tais que um jogador tem três símbolos em uma linha ou todos os quadrados são preenchidos. O número em cada nó de folha indica o valor de utilidade do estado terminal, do ponto de vista de MAX; valores altos são considerados bons para MAX e ruins para MIN (o que explica os nomes dados aos jogadores).

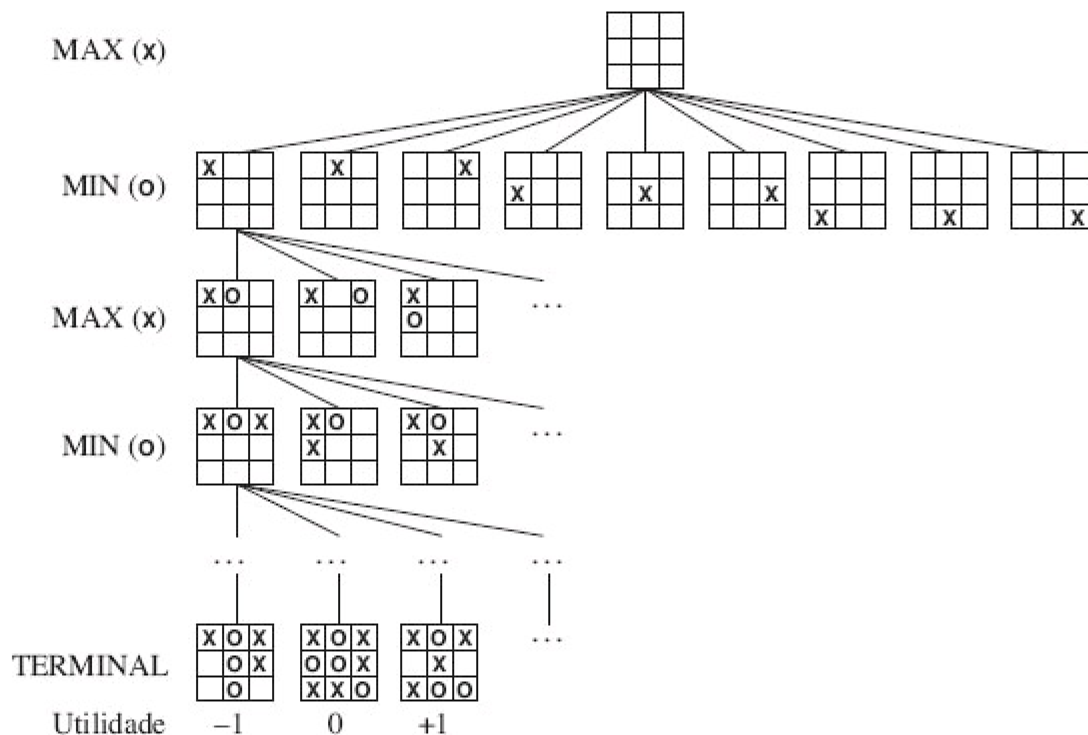


Figura 5.1 Uma árvore de busca (parcial) para o jogo da velha. O nó superior é o estado inicial, e MAX faz o primeiro movimento colocando um X em um quadrado vazio. Mostramos parte da árvore de busca fornecendo movimentos alternados por MIN (O) e MAX(x), até alcançarmos finalmente os estados terminais, aos quais podem ser atribuídas utilidades de acordo com as regras do jogo.

Para o jogo da velha, a árvore de jogo é relativamente pequena, menos de 9! = 362.880 nós terminais. Mas, para o xadrez, há mais de 10^{40} nós, de modo que é melhor pensar na árvore de jogo como sendo uma construção teórica que não podemos perceber no mundo físico. Mas, independentemente do tamanho da árvore de jogo, é trabalho de MAX a busca de uma boa jogada. Usamos o termo **árvore de busca** para uma árvore que está sobreposta à árvore de jogo completa, examinando os nós o suficiente para permitir que um jogador determine que lance fazer.

5.2 DECISÕES ÓTIMAS EM JOGOS

Em um problema de busca normal, a solução ótima seria uma sequência de ações que levasse a um estado objetivo — um estado terminal que representa uma vitória. Por outro lado, em um jogo, MIN tem alguma relação com esse estado. Portanto, MAX deve encontrar uma **estratégia** de contingência

que especifique o movimento de MAX no estado inicial e depois os movimentos de MAX nos estados resultantes de cada resposta possível de MIN, e depois os movimentos de MAX nos estados resultantes de cada resposta possível de MIN a *esses* movimentos, e assim por diante.

Isso é exatamente análogo ao algoritmo E-OU de busca (Figura 4.11), com MAX no papel de OU e MIN equivalente a E. Grosseiramente falando, uma ótima estratégia leva a resultados pelo menos tão bons como qualquer outra estratégia quando se está jogando com um adversário infalível. Começaremos mostrando como encontrar essa estratégia ótima.

Até mesmo um jogo simples como o jogo da velha é muito complexo para traçarmos a árvore de jogo inteira em uma página e, assim, nos limitaremos ao jogo trivial da Figura 5.2. Os movimentos possíveis para MAX no nó raiz são identificados por a_1, a_2 e a_3. As respostas possíveis para a_1 correspondentes a MIN são b_1, b_2 e b_3, e assim sucessivamente. Esse jogo específico termina depois de um movimento realizado por MAX e por MIN. (No linguajar dos jogos, dizemos que essa árvore tem a profundidade de um único movimento, que consiste em dois meios movimentos, cada um dos quais é chamado **jogada**.) As utilidades dos estados terminais nesse jogo variam de 2 a 14.

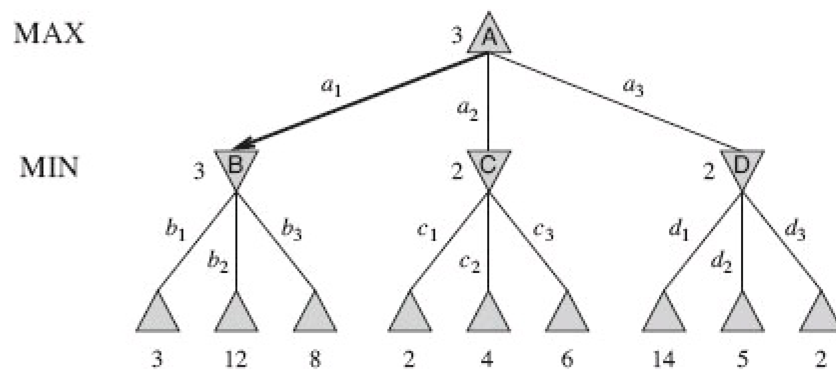


Figura 5.2 Uma árvore de jogo de duas jogadas. Os nós Δ são "nós de MAX", nos quais é a vez de MAX efetuar um movimento, e os nós ∇ são "nós de MIN". Os nós terminais mostram os valores de utilidade para MAX;os outros nós estão identificados com seus valores minimax. O melhor movimento de MAX na raiz é a_1 porque leva a um estado com o mais alto valor minimax, e a melhor resposta de MIN é b_1 porque leva a um estado com o mais baixo valor minimax.

Dada uma árvore de jogo, a estratégia ótima pode ser determinada do **valor minimax** de cada nó, que representamos como VALOR-MINIMAX(n). O valor minimax de um nó é a utilidade (para MAX) de se encontrar no estado correspondente, *supondo-se que ambos os jogadores tenham desempenho ótimo* desde esse estado até o fim do jogo. É óbvio que o valor minimax de um estado terminal é simplesmente sua utilidade. Além disso, dada uma escolha, MAX preferirá se mover para um estado de valor máximo, enquanto MIN preferirá um estado de valor mínimo. Assim, temos:

$$\text{VALOR-MINIMAX}(s) = \begin{cases} \text{UTILIDADE}(s) & \text{se TESTE DE TÉRMINO}(s) \\ \text{Max}_{a \in Ações(s)} \text{ MINIMAX (RESULTADO}(s,a)) & \text{se JOGADOR}(s) = \text{MAX} \\ \text{Min}_{a \in Ações(s)} \text{ MINIMAX (RESULTADO}(s,a)) & \text{se JOGADOR}(s) = \text{MIN} \end{cases}$$

Vamos aplicar essas definições à árvore de jogo da Figura 5.2. Os nós terminais no nível inferior obtiverem os valores utilidade da função UTILIDADE do jogo. O primeiro nó de MIN, identificado

por *B*, tem três sucessores com valores 3, 12 e 8; portanto, seu valor minimax é 3. De modo semelhante, os outros dois nós de MIN têm valor minimax 2. O nó raiz é um nó de MAX; seus estados sucessores têm valores minimax 3, 2 e 2; logo, ele tem um valor minimax igual a 3. Também podemos identificar a **decisão minimax** na raiz: a ação a_1 é a escolha ótima para MAX porque leva ao estado com o mais alto valor minimax.

Essa definição de jogo ótimo para MAX supõe que MIN também jogue de forma ótima — ela maximiza o resultado para MAX no *pior caso*. E se MIN não jogar de forma ótima? Nesse caso, é fácil mostrar (Exercício 5.7) que MAX terá um desempenho ainda melhor. Pode haver outras estratégias contra oponentes não ótimos que poderão funcionar melhor que a estratégia de minimax; porém, essas estratégias necessariamente têm um desempenho pior contra oponentes ótimos.

5.2.1 O algoritmo minimax

O **algoritmo minimax** (Figura 5.3) calcula a decisão minimax a partir do estado corrente. Ela utiliza uma computação recursiva simples dos valores minimax de cada estado sucessor, implementando diretamente as equações da definição. A recursão percorre todo o caminho descendente até as folhas da árvore e, depois, os valores minimax são **propagados de volta** pela árvore, à medida que a recursão retorna. Por exemplo, na Figura 5.2, primeiro o algoritmo efetua uma recursão descendo a árvore até os três nós de folhas inferiores e emprega a função UTILIDADE sobre eles para descobrir que seus valores são 3, 12 e 8, respectivamente. Em seguida, ele toma o mínimo desses valores, 3, e o devolve como valor propagado de volta para o nó *B*. Um processo semelhante fornece os valores propagados de volta de 2 para *C* e 2 para *D*. Por fim, tomamos o valor máximo entre 3, 2 e 2 para obter o valor propagado de volta igual a 3 para o nó raiz.

```
função DECISÃO-MINIMAX(estado) retorna uma ação
   retornar arg maxa∈AçõES(s) VALOR-MIN(RESULTADO(estado, a))

função VALOR-MAX(estado) retorna um valor de utilidade
   se TESTE TERMINAL (estado) então retornar UTILIDADE(estado)
   v ← − ∞
   para cada a em AÇÕES(estado) faça
      v ← MAX(v,VALOR-MIN(RESULTADO(s,a)))
   retornar v

função VALOR-MIN(estado) retorna um valor de utilidade
   se TESTE-TERMINAL(estado) então retornar UTILIDADE(estado)
   v ← − ∞
   para cada a em AÇÕES(estado) faça
      v ← MIN(v,VALOR-MAX(RESULTADO(s, a)))
   retornar v
```

Figura 5.3 Um algoritmo para calcular decisões minimax. Ele retorna a ação correspondente ao melhor movimento possível, isto é, o movimento que leva ao resultado com a melhor utilidade, sob a suposição de que o oponente joga para minimizar a utilidade. As funções VALOR-MAX e VALOR-MIN passam por toda a árvore de jogo, até chegar às folhas, a fim de determinar o valor de propagação de volta de um estado. A notação argmax$_{a \in S}$ $f(a)$ calcula o elemento a do conjunto S que possui o valor máximo de $f(a)$.

O algoritmo minimax executa uma exploração completa em profundidade da árvore de jogo. Se a profundidade máxima da árvore é m e existem b movimentos válidos em cada ponto, a complexidade de tempo do algoritmo minimax é $O(b^m)$. A complexidade de espaço é $O(b^m)$ para um algoritmo que gera todos os sucessores de uma vez ou $O(m)$ para um algoritmo que gera ações, uma de cada vez. É claro que, em jogos reais, o custo de tempo é totalmente impraticável, mas esse algoritmo serve como base para a análise matemática de jogos e para algoritmos mais práticos.

5.2.2 Decisões ótimas em jogos com vários participantes

Muitos jogos populares permitem mais de dois jogadores. Vamos examinar a maneira de estender a ideia de minimax a jogos com vários jogadores. Isso é simples do ponto de vista técnico, mas destaca algumas questões conceituais novas e interessantes.

Primeiro, precisamos substituir o único valor para cada nó por um *vetor* de valores. Por exemplo, em um jogo de três jogadores com os participantes *A*, *B* e *C*, um vetor $\langle v_A, v_B, v_C \rangle$ está associado a cada nó. Para os estados terminais, esse vetor fornece a utilidade do estado do ponto de vista de cada jogador. (Em jogos de soma zero com dois jogadores, o vetor de dois elementos pode ser reduzido a um único valor porque os valores são sempre opostos.) O caminho mais simples para implementar isso é fazer a função UTILIDADE retornar um vetor de utilidades.

Agora temos de considerar os estados não terminais. Vamos examinar o nó identificado por *X* na árvore de jogo da Figura 5.4. Nesse estado, o jogador *C* define o que fazer. As duas escolhas levam a estados terminais com vetores de utilidade $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$ e $\langle v_A = 4, v_B = 2, v_C = 3 \rangle$. Tendo em vista que 6 é maior que 3, *C* deve-se escolher o primeiro movimento. Isso significa que, se o estado *X* for alcançado, a jogada subsequente levará a um estado terminal com utilidades $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$. Consequentemente, o valor de *X* que foi propagado de volta é esse vetor. Em geral, o valor propagado de volta de um nó *n* é sempre o vetor de utilidade do estado do sucessor com o mais alto valor para a escolha do jogador em *n*. Qualquer pessoa que participa de jogos com vários jogadores, como Diplomacy™, logo fica ciente de que muito mais acontece do que em jogos de dois jogadores. Os jogos com vários participantes normalmente envolvem **alianças**, sejam elas formais ou informais, entre os jogadores. As alianças são feitas e desfeitas à medida que o jogo se desenrola. Como entender tal comportamento? As alianças constituem uma consequência natural de estratégias ótimas para cada jogador em um jogo com vários participantes? É possível que sim. Por exemplo, vamos supor que *A* e *B* estejam em posições fracas e que *C* esteja em uma posição mais forte. Então, com frequência, é ótimo para *A* e *B* atacarem *C* em vez de atacarem um ao outro, para que *C* não destrua cada um deles individualmente. Desse modo, a colaboração emerge de um comportamento

puramente egoísta. É claro que, tão logo *C* se enfraqueça sob o violento ataque conjunto, a aliança perderá seu valor, e *A* ou *B* poderá violar o acordo. Em alguns casos, as alianças explícitas apenas tornam concreto aquilo que teria acontecido de qualquer modo. Em outros casos, um estigma social incorpora-se para romper uma aliança, de forma que os jogadores devem buscar o equilíbrio entre a vantagem imediata de romper uma aliança e a desvantagem a longo prazo de serem considerados pouco confiáveis. Veja a Seção 17.5 para obter mais informações sobre essas complicações.

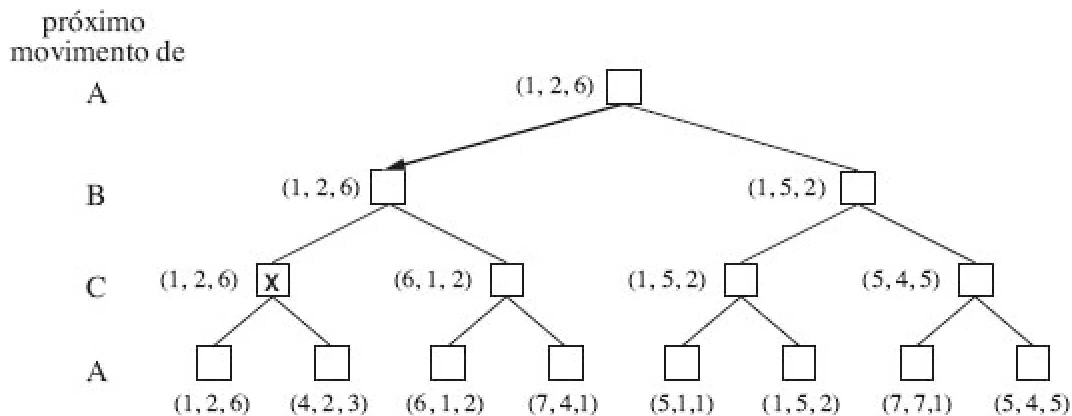


Figura 5.4 As três primeiras jogadas de uma árvore de jogo com três jogadores (*A*, *B*, *C*). Cada nó é identificado com valores do ponto de vista de cada jogador. O melhor movimento está assinalado na raiz.

Se o jogo for de soma diferente de zero, a colaboração também poderá ocorrer com apenas dois jogadores. Por exemplo, vamos supor que exista um estado terminal com utilidades $\langle v_A = 1.000, v_B = 1.000\rangle$ e que 1.000 seja a mais alta utilidade possível para cada jogador. Então, a estratégia ótima é a de ambos os jogadores fazerem todo o possível para alcançar esse estado, isto é, os jogadores cooperarão de forma automática para atingir uma meta mutuamente desejável.

5.3 PODA ALFA-BETA

O problema da busca minimax é que o número de estados de jogo que ela tem de examinar é exponencial em relação ao número de movimentos. Infelizmente, não podemos eliminar o expoente, mas resulta que podemos efetivamente reduzi-lo pela metade. O artifício é a possibilidade de calcular a decisão minimax correta sem examinar todos os nós na árvore de jogo. Ou seja, podemos tomar emprestada a ideia de **poda** do Capítulo 3, a fim de poder deixar de considerar grandes partes da árvore. A técnica específica que examinaremos é chamada **poda alfa-beta**. Quando é aplicada a uma árvore minimax padrão, ela retorna o mesmo movimento que minimax retornaria, mas poda as ramificações que não terão influência possível sobre a decisão final.

Considere novamente a árvore de jogo de duas jogadas da Figura 5.2. Vamos acompanhar mais uma vez o cálculo da decisão ótima, agora prestando bastante atenção ao que conhecemos em cada ponto do processo.

Os passos são explicados na Figura 5.5. O resultado é que podemos identificar a decisão minimax sem jamais avaliar dois dentre os nós de folhas.

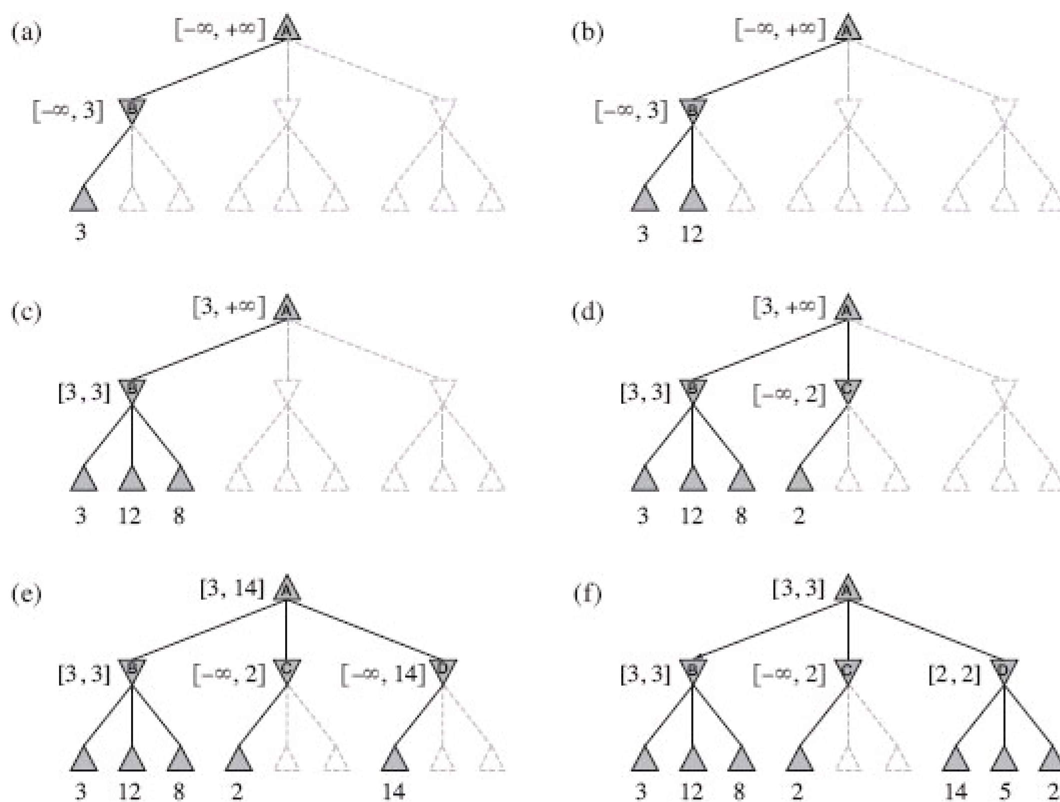


Figura 5.5 Fases no cálculo da decisão ótima para a árvore de jogo da Figura 5.2. Em cada ponto, mostramos o intervalo de valores possíveis para cada nó. (a) Aprimeira folha sob *B* tem valor 3. Consequentemente, *B*, que é um nó de MIN, tem valor *máximo* 3. (b) A segunda folha sob *B* tem valor 12; MIN evitaria esse movimento, de forma que o valor de *B* ainda é, no máximo, 3. (c) Aterceira folha sob *B* tem valor 8; vimos todos os estados sucessores de *B* e, assim, o valor de *B* é exatamente 3. Agora, podemos deduzir que o valor da raiz é *pelo menos* 3, porque MAX tem uma escolha de valor 3 na raiz. (d) A primeira folha abaixo de *C* tem o valor 2. Consequentemente, *C*, que é um nó de MIN, tem valor *máximo* 2. Porém, sabemos que *B* vale 3; portanto, MAX nunca escolheria *C*. Desse modo, não há razão para se examinar os outros sucessores de *C*. Esse é um exemplo de poda alfa-beta. (e) A primeira folha abaixo de *D* tem o valor 14, e então *D* vale *no máximo* 14. Esse valor ainda é mais alto que a melhor alternativa de MAX (isto é, 3) e, portanto, precisamos continuar a explorar sucessores de *D*. Note também que agora temos limites para todos os sucessores da raiz e, consequentemente, o valor da raiz também é no máximo 14. (f) O segundo sucessor de *D* vale 5 e, assim, novamente precisamos continuar a exploração. O terceiro sucessor vale 2; agora, *D* vale exatamente 2. A decisão de MAX na raiz é efetuar o movimento para *B*, o que nos dá o valor 3.

Isso também pode ser visto como uma simplificação da fórmula de VALOR-MINIMAX. Sejam x e y valores dos dois sucessores não avaliados do nó *C* na Figura 5.5 e seja z o mínimo entre x e y. Então, o valor do nó raiz é dado por:

$$\begin{aligned}\text{MINIMAX}(raiz) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \qquad \text{onde } z = \min(2, x, y) \leq 2 \\ &= 3.\end{aligned}$$

Em outras palavras, o valor da raiz e, consequentemente, a decisão minimax são *independentes* dos valores das folhas podadas x e y.

A poda alfa-beta pode ser aplicada a árvores de qualquer profundidade e frequentemente é possível podar subárvores inteiras em lugar de podar apenas folhas. O princípio geral é este: considere um nó *n* em algum lugar na árvore (veja a Figura 5.6), tal que o Jogador tenha a escolha de movimento até esse nó. Se o Jogador tiver uma escolha melhor *m* no nó pai de *n* ou em qualquer ponto de escolha adicional acima dele, então *n nunca será alcançado em um jogo real*. Assim, uma vez que descobrimos o suficiente sobre *n* (examinando alguns de seus descendentes) para chegar a essa conclusão, poderemos podá-lo.

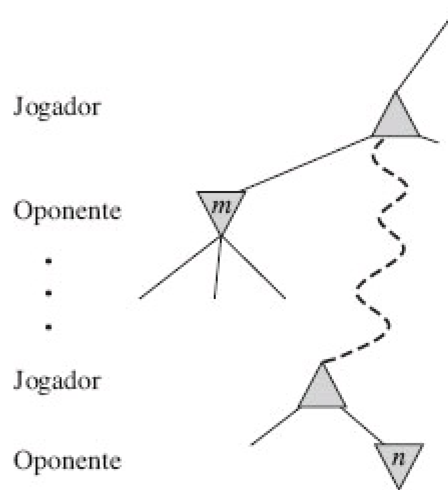


Figura 5.6 O caso geral de poda alfa-beta. Se *m* é melhor que *n* para o Jogador, nunca chegaremos a *n* em um jogo.

Lembre-se de que a busca minimax é do tipo em profundidade; então, em qualquer instante só temos de considerar os nós ao longo de um único caminho na árvore. A poda alfa-beta obtém seu nome a partir dos dois parâmetros a seguir, que descrevem limites sobre os valores propagados de volta que aparecem em qualquer lugar ao longo do caminho:

α = o valor da melhor escolha (isto é, a de valor mais alto) que encontramos até o momento em qualquer ponto de escolha ao longo do caminho para MAX.
β = o valor da melhor escolha (isto é, a de valor mais baixo) que encontramos até agora em qualquer ponto de escolha ao longo do caminho para MIN.

A busca alfa-beta atualiza os valores de α e β à medida que prossegue e poda as ramificações restantes em um nó (isto é, encerra a chamada recursiva) tão logo se sabe que o valor do nó corrente é pior que o valor corrente de α ou β para MAX ou MIN, respectivamente. O algoritmo completo é mostrado na Figura 5.7. Encorajamos o leitor a acompanhar seu comportamento quando ele é aplicado à árvore da Figura 5.5.

função BUSCA-ALFA-BETA(*estado*) **retorna** uma ação
  v ← VALOR-MAX(*estado*, –∞, +∞)
  retornar a *ação* em AÇÕES(*estado*) com valor *v*

função VALOR-MAX(*estado*, α, β) **retorna** *um valor de utilidade*
  se TESTE-TERMINAL(*estado*) **então retornar** UTILIDADE(*estado*)

```
  v← −∞
  para cada a, em AÇÕES(estado) faça
    v ← MAX(v, VALOR-MIN(RESULTADO(s, a), α, β))
    se v ≥ β então retornar v
    α ← MAX(α, v)
  retornar v
```

```
função VALOR-MIN(estado, α, β) retorna um valor de utilidade
  se TESTE-TERMINAL(estado) então retornar UTILIDADE(estado)
  v ← +∞
  para cada a, em AÇÕES(estado) faça
    v ← MIN(v, VALOR-MIN(RESULTADO(s, a), α, β))
    se v ≤ a então retornar v
    β ← MIN(β, v)
  retornar v
```

Figura 5.7 O algoritmo de busca alfa-beta. Note que essas rotinas são idênticas às rotinas de MINIMAX da Figura 5.3, com exceção das duas linhas em cada uma das funções VALOR-MIN e VALOR-MAX que mantêm α e β (e da necessidade de repassar esses parâmetros).

5.3.1 Ordenação de movimentos

A efetividade da poda alfa-beta é altamente dependente da ordem em que os estados são examinados. Por exemplo, na Figura 5.5(e) e (f), não poderíamos podar quaisquer sucessores de *D* porque os piores sucessores (do ponto de vista de MIN) foram gerados primeiro. Se o terceiro sucessor tivesse sido gerado primeiro, seríamos capazes de podar os outros dois. Isso sugere que poderia valer a pena tentar examinar primeiro os sucessores que têm probabilidade de serem melhores.

Se supusermos que isso pode ser feito,[2] então o resultado será que alfa-beta precisará examinar apenas $O(b^{m/2})$ nós para escolher o melhor movimento, em vez de $O(b^m)$ para minimax. Isso significa que o fator de ramificação efetivo se tornará $\sqrt{b}$ em vez de *b* — no caso do xadrez, 6 em vez de 35. Em outras palavras, alfa-beta poderá resolver uma árvore aproximadamente duas vezes tão profunda como minimax no mesmo período de tempo. Se os sucessores forem examinados em ordem aleatória, em vez de se tomar o melhor em primeiro lugar, o número total de nós examinados será cerca de $O(b^{3m/4})$ para um valor moderado de *b*. No caso do xadrez, uma função de ordenação bastante simples (como experimentar capturas primeiro, depois ameaças, depois movimentos para a frente e, em seguida, movimentos para trás) levará você a uma distância de aproximadamente duas vezes o resultado do melhor caso, $O(b^{m/2})$.

Acrescentar esquemas dinâmicos de ordenação de movimentos, como tentar primeiro os movimentos considerados os melhores da última vez, nos levará até bem perto do limite teórico. O passado pode ser o lance anterior — muitas vezes, as mesmas ameaças permanecem — ou poderia

vir da exploração do lance atual. Uma maneira de obter informações do lance atual é com busca de aprofundamento iterativo da pesquisa. Primeiro, pesquise uma jogada profunda e registre o melhor caminho de movimentos. Em seguida, busque uma jogada mais profunda, mas utilize o caminho registrado para informar a ordenação do movimento. Como vimos no Capítulo 3, o aprofundamento iterativo em uma árvore de jogo exponencial acrescenta apenas uma fração constante para o tempo total de busca, que pode ser mais do que compensado por uma melhor ordenação de movimento. As melhores jogadas são muitas vezes chamadas de **lances mortais** e tentá-los de primeira é chamado de heurística de lance mortal.

No Capítulo 3, observamos que estados repetidos na árvore de busca podem causar um aumento exponencial no custo da busca. Em muitos jogos, estados repetidos ocorrem com frequência devido a **transposições** — permutações diferentes da mesma sequência que terminam na mesma posição. Por exemplo, se as brancas têm um movimento a_1 que pode ser respondido pelas pretas com b_1 e um movimento não relacionado a_2 no outro lado do tabuleiro que pode ser respondidos por b_2, as sequências $[a_1, b_1, a_2, b_2]$ e $[a_1, b_2, a_2, b_1]$ terminarão na mesma posição. Vale a pena armazenar a avaliação dessa posição resultante em uma tabela de hash na primeira vez em que ela for encontrada, de forma que não tenhamos de recalculá-la em ocorrências subsequentes.

A tabela de hash de posições já vistas é tradicionalmente chamada **tabela de transposição**; em essência, ela é idêntica à lista *explorada* em BUSCA-EM-GRAFO (Seção 3.3). O uso de uma tabela de transposição pode ter um efeito drástico, chegando às vezes a duplicar a profundidade de busca acessível no xadrez. Por outro lado, se estivermos avaliando um milhão de nós por segundo, não será prático manter *todos* eles na tabela de transposição. São usadas diversas estratégias para escolher os nós que devem ser mantidos e os que devem ser descartados.

5.4 DECISÕES IMPERFEITAS EM TEMPO REAL

O algoritmo minimax gera o espaço de busca do jogo inteiro, enquanto o algoritmo alfa-beta nos permite podar grandes partes desse espaço. Porém, alfa-beta ainda tem de fazer a busca em toda a distância até os estados terminais, pelo menos para uma parte do espaço de busca. Em geral, essa profundidade não é prática porque os movimentos devem ser realizados em um período de tempo razoável — normalmente por alguns minutos, no máximo. O artigo de Claude Shannon, *Programming a computer for playing chess* (1950), propunha em vez disso que os programas cortassem a busca mais cedo e aplicassem uma **função de avaliação** heurística aos estados da busca, transformando efetivamente nós não terminais em folhas terminais. Em outras palavras, a sugestão é alterar minimax ou alfa-beta de duas maneiras: substituir a função utilidade por uma função de avaliação de heurística AVAL, que fornece uma estimativa da utilidade da posição, e o teste de término por um **teste de corte** que decide quando aplicar AVAL. Isso nos dá o seguinte para minimax heurística para o estado s e profundidade máxima d:

$$\text{H-MINIMAX}(s, d) = \begin{cases} \text{AVAL}(s) & \text{se TESTE DE CORTE}(s, d) \\ \max_{a \in \text{Ações}(s)} \text{H-MINIMAX}(\text{RESULTADO}(s, a), d + 1) & \text{se JOGADOR}(s) = \text{MAX} \\ \min_{a \in \text{Ações}(s)} \text{H-MINIMAX}(\text{RESULTADO}(s, a), d + 1) & \text{se JOGADOR}(s) = \text{MIN.} \end{cases}$$

5.4.1 Funções de avaliação

Uma função de avaliação retorna uma *estimativa* da utilidade esperada do jogo, a partir de uma dada posição, da mesma forma que as funções de heurísticas do Capítulo 3 retornam uma estimativa da distância até a meta.

A ideia de um avaliador não era nova quando Shannon a propôs. Durante séculos, os jogadores de xadrez (e os aficionados por outros jogos) desenvolveram meios de julgar o valor de uma posição porque os seres humanos são ainda mais limitados que os programas de computador no volume de busca que podem realizar. Deve ficar claro que o desempenho de um programa de jogos depende fortemente da qualidade de sua função de avaliação. Uma função de avaliação inexata guiará um agente em direção a posições que acabarão por ser perdidas. Qual é a maneira exata de projetarmos boas funções de avaliação?

Primeiro, a função de avaliação deve ordenar os estados *terminais* do mesmo modo que a verdadeira função utilidade: estados que são vitórias que devem avaliar empates, que por sua vez devem ser melhores que perdas. Caso contrário, um agente que utilizasse a função de avaliação poderia errar, mesmo que pudesse antecipar todos os movimentos até o fim do jogo. Em segundo lugar, a computação não deve demorar tempo demais! (O ponto fundamental é a busca mais rápida.) Em terceiro lugar, no caso de estados não terminais, a função de avaliação deve estar fortemente relacionada com as chances reais de vitória.

O leitor deve ter se surpreendido com a expressão "chances de vitória". Afinal, o xadrez não é um jogo de azar: conhecemos o estado corrente com certeza e não há dados envolvidos no processo. Contudo, se a busca tiver de ser cortada em estados não terminais, o algoritmo será necessariamente *incerto* sobre os resultados finais desses estados. Esse tipo de incerteza é induzido por limitações computacionais, não informativas.

Dado o volume limitado de computação que a função de avaliação pode realizar para determinado estado, o melhor que ela pode fazer é arriscar um palpite sobre o resultado final.

Vamos tornar essa ideia mais concreta. A maioria das funções de avaliação atua calculando diversas **características** do estado — por exemplo, no xadrez, teríamos características para o número de peões brancos, pretos, rainhas brancas, pretas, e assim por diante. Consideradas em conjunto, as características definem diversas *categorias* ou *classes de equivalência* de estados: os estados de cada categoria têm os mesmos valores para todas as características. Por exemplo, ao final do jogo, uma categoria contém ao todo dois peões *versus* um peão. Qualquer categoria específica, em termos gerais, conterá alguns estados que levam a vitórias, alguns que levam a empates e alguns que levam a derrotas. A função de avaliação não tem como saber quais são os estados de cada grupo, mas pode retornar um único valor capaz de refletir a *proporção* de estados que conduzem a cada resultado. Por exemplo, vamos supor que nossa experiência sugira que 72% dos estados encontrados na categoria dois peões *versus* um peão levem a uma vitória (com utilidade +1); 20% levem a uma derrota (0) e 8% a um empate (1/2). Então, uma avaliação razoável dos estados na categoria é a média ponderada ou o **valor esperado**: $(0,72 \times +1) + (0,20 \times 0) + (0,08 \times 1/2) = 0,76$. Em princípio, o valor esperado pode ser determinado para cada categoria, o que resulta em uma função de avaliação que funciona para qualquer estado. Como ocorre com estados terminais, a função de avaliação não precisa retornar valores esperados reais, desde que a *ordenação* dos estados seja a

mesma.

Na prática, essa espécie de análise exige muitas categorias e, consequentemente, muita experiência para estimar todas as probabilidades de vitória. Em vez disso, a maioria das funções de avaliação calcula contribuições numéricas separadas de cada característica e depois as *combina* para encontrar o valor total. Por exemplo, os livros introdutórios de xadrez fornecem um **valor material** aproximado para cada peça: cada peão vale 1, um cavalo ou um bispo pena 3, uma torre 5 e a rainha 9. Outras características, como "boa estrutura de peões" e "segurança do rei" poderiam valer, digamos, metade de um peão. Esses valores de características são então simplesmente somados para se obter a avaliação da posição.

Uma vantagem segura equivalente a um peão fornece uma probabilidade substancial de vitória, e uma vantagem segura equivalente a três peões deve proporcionar uma vitória quase certa, como ilustra a Figura 5.8(a). Matematicamente, essa espécie de função de avaliação é chamada **função linear ponderada** porque pode ser expressa como:

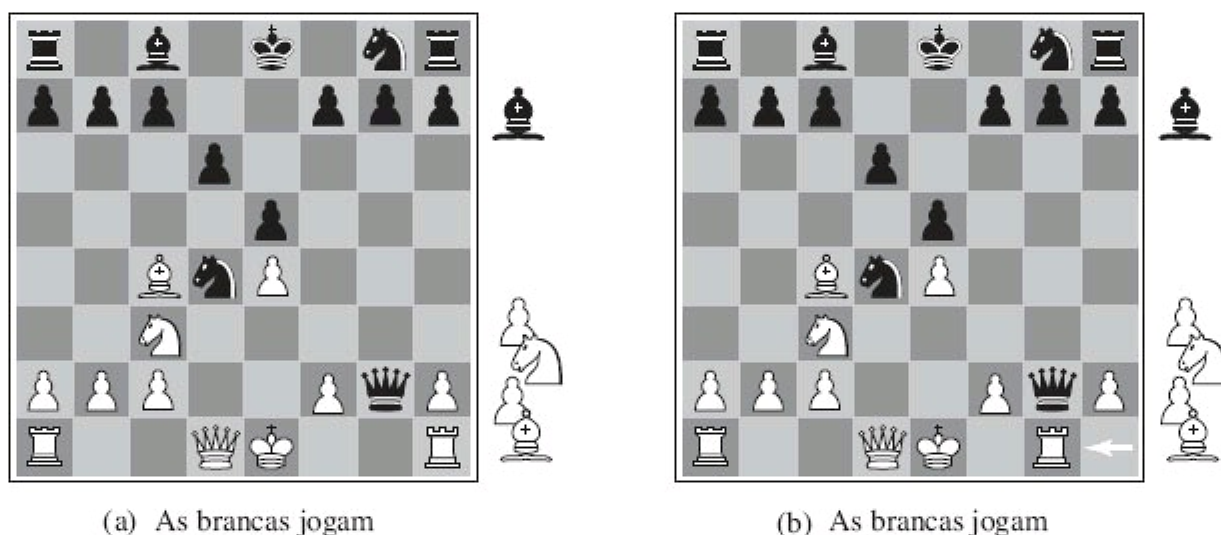


Figura 5.8 Duas posições de xadrez, que diferem apenas na posição da torre na parte inferior direita. Em (a), as pretas têm uma vantagem de um cavalo e dois peões, que deveria ser o suficiente para vencer o jogo. Em (b), a branca vai capturar a rainha, dando-lhe uma vantagem que deveria ser forte o suficiente para vencer.

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s) ,$$

onde cada w_i é um peso e cada f_i é uma característica da posição. No caso do xadrez, f_i poderia representar os números de cada tipo de peça no tabuleiro, w_i poderia corresponder aos valores das peças (1 para peão, 3 para bispo etc.).

Somar os valores de características parece algo razoável, mas, na verdade, envolve uma suposição muito forte: que a contribuição de cada característica é *independente* dos valores das outras características. Por exemplo, a atribuição do valor 3 a um bispo ignora o fato de que os bispos são mais poderosos no fim do jogo, quando têm bastante espaço de manobra.

Por essa razão, os programas atuais de xadrez e de outros jogos também utilizam combinações *não lineares* de características. Por exemplo, um par de bispos poderia valer um pouco mais que o dobro

do valor de um único bispo, e um bispo poderia valer mais no fim do jogo que no início (isto é, quando a característica do *número de lances* for alta ou a característica do *número de peças restantes* for baixa).

O leitor atento notará que as características e os pesos *não* fazem parte das regras do xadrez! Eles vêm de séculos de experiência humana no jogo de xadrez. Em jogos nos quais esse tipo de experiência não está disponível, os pesos da função de avaliação podem ser estimados pelas técnicas de aprendizado de máquina do Capítulo 18. Vale a pena reafirmar que a aplicação dessas técnicas ao xadrez confirma que um bispo vale realmente cerca de três peões.

5.4.2 Busca com corte

A próxima etapa é modificar BUSCA-ALFA-BETA, de modo que ela chame a função heurística AVAL quando for apropriado cortar a busca. Em termos de implementação, substituímos as duas linhas da Figura 5.7 que mencionam TESTE-TERMINAL pela linha a seguir:

se TESTE-DE-CORTE(*estado*, *profundidade*) **então retornar** AVAL(*estado*).

Também devemos providenciar alguma notação para que a *profundidade* corrente seja incrementada em cada chamada recursiva. A abordagem mais direta para controlar a quantidade de busca é definir um limite de profundidade fixo, a fim de que TESTE-DE-CORTE (*estado*, *profundidade*) retorne *verdadeiro* para toda *profundidade* maior que alguma profundidade fixa *d* (ela também deve retornar *verdadeiro* para todos os estados terminais, como fazia TESTE-TERMINAL). A profundidade *d* é escolhida de modo que um movimento seja selecionado dentro do tempo previsto. Uma abordagem mais resistente é aplicar o aprofundamento iterativo (veja o Capítulo 3). Quando o tempo se esgota, o programa retorna o movimento selecionado pela busca mais profunda concluída. Como bônus, o aprofundamento iterativo também ajuda com a ordenação do movimento.

No entanto, essas simples abordagens podem levar a erros, devido à natureza aproximada da função de avaliação. Considere mais uma vez a função de avaliação simples para xadrez, baseada na vantagem material. Suponha que o programa pesquise até a profundidade limite, alcançando a posição da Figura 5.8(b), onde as pretas têm a vantagem de um cavalo e dois peões. Isso seria reportado como o valor heurístico do estado, declarando-se assim que o estado será uma vitória provável das peças pretas. Porém, o próximo movimento das brancas captura a rainha preta sem qualquer compensação. Portanto, a posição resulta na realidade em uma vitória das brancas, mas isso só pode ser visto observando-se mais uma jogada à frente.

É óbvio que é necessário um teste de corte mais sofisticado. A função de avaliação deve ser aplicada apenas a posições **quiescentes** — isto é, posições em que é improvável haver grandes mudanças de valores no futuro próximo. Por exemplo, no xadrez, as posições em que podem ser feitas capturas favoráveis não são quiescentes para uma função de avaliação que simplesmente efetua a contagem material. Posições não quiescentes podem ser expandidas adiante, até serem alcançadas posições quiescentes. Essa busca extra é chamada de **busca de quiescência**; às vezes, ela se restringe a considerar apenas certos tipos de movimentos, como movimentos de captura, que resolverão com

rapidez as incertezas da posição.

O **efeito de horizonte** é mais difícil de eliminar. Ele surge quando o programa está enfrentando um movimento feito pelo oponente que causa sérios danos e, em última instância, inevitável, mas poderia ser evitado temporariamente, através de táticas de adiamento. Considere o jogo de xadrez na Figura 5.9. É claro que não há caminho para o bispo preto fugir. Por exemplo, a torre branca pode capturá-lo movendo-se para h1, depois a1, depois a2; em seguida, uma captura à profundidade da jogada 6. Mas as pretas têm uma sequência de movimentos que empurra a captura do bispo "além do horizonte". Suponha que os pretas busquem a profundidade da jogada 8. A maioria dos movimentos das pretas vai levar à captura eventual do bispo e, portanto, serão marcadas como lances "ruins". Mas as pretas considerarão o xeque no rei branco com o peão em e4. Isso fará com que o rei capture o peão. Agora as pretas vão considerar o xeque novamente, com o peão em f5, levando a outra captura de peão. Isso leva quatro jogadas, e as quatro jogadas restantes não são suficientes para capturar o bispo. As pretas pensam que a linha de jogo poupou o bispo, ao preço de dois peões, quando na verdade tudo o que foi feito é empurrar a inevitável captura do bispo para além do horizonte que as pretas podiam visualizar.

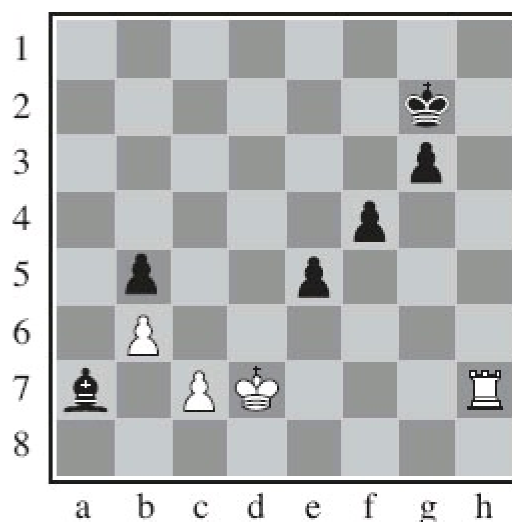


Figura 5.9 O efeito de horizonte. Com as pretas se movendo, o bispo preto está certamente condenado. Mas as pretas podem evitar esse evento marcando o rei branco com seus peões, obrigando o rei a capturar os peões. Isso empurra a perda inevitável do bispo sobre o horizonte e, portanto, o sacrifício dos peões é visto pelo algoritmo de busca como boas jogadas e não como más.

Uma estratégia para mitigar o efeito horizonte é a **extensão singular**, um movimento que é "claramente melhor" do que todos os outros movimentos em determinada posição. Uma vez descoberto em qualquer lugar da árvore no curso de uma busca, esse movimento singular é lembrado. Quando a busca atinge o limite de profundidade normal, o algoritmo verifica se a extensão singular é um movimento legal; se for, permite que o lance seja considerado. Isso faz com que a árvore fique mais profunda, mas, havendo poucas extensões singulares, não adicionará muitos nós totais à árvore.

5.4.3 Poda adiantada

Até agora, mencionamos a busca com corte em certo nível e dissemos que a realização da busca

alfa-beta não tem nenhum efeito comprovado sobre o resultado (pelo menos com respeito aos valores de avaliação heurística). Também é possível efetuar a **poda adiantada**, significando que alguns movimentos em dado nó serão podados de imediato, sem consideração adicional. É claro que a maioria dos seres humanos que jogam xadrez só considera alguns movimentos a partir de cada posição (pelo menos de forma consciente). Uma abordagem à poda adiantada é a **busca em feixe**: em cada jogada, considera-se apenas um "feixe" das n melhores jogadas (de acordo com a função de avaliação) em vez de considerar todos os movimentos possíveis. Infelizmente, a abordagem é bastante perigosa porque não há nenhuma garantia de que o melhor movimento não será podado.

O algoritmo PROBCUT, ou corte probabilístico (Buro, 1995), é uma versão da poda adiantada de busca alfa-beta que utiliza a estatística adquirida com a experiência prévia para diminuir a chance de a melhor jogada ser podada. A busca alfa-beta poda qualquer nó que esteja *provavelmente* fora da janela (α, β) atual. O PROBCUT também poda nós que *provavelmente* estão fora da janela. Ela calcula essa probabilidade fazendo uma busca superficial para calcular o valor propagado v de um nó e, em seguida, usa a experiência passada para estimar como é que a pontuação de v à profundidade d na árvore ficaria fora de (α, β). Buro aplicou essa técnica ao programa Othello, LOGISTELLO, e descobriu que uma versão de seu programa com PROBCUT bateu a versão regular em 64% do tempo, mesmo quando se dá o dobro de tempo à versão regular.

A combinação de todas as técnicas descritas aqui resulta em um programa que pode jogar xadrez (ou outros jogos) de modo respeitável. Vamos supor que implementamos uma função de avaliação para xadrez, um teste de corte razoável com uma busca de quiescência, e ainda uma grande tabela de transposição. Vamos supor também que, depois de meses de tediosa escovação de bits, podemos gerar e avaliar cerca de um milhão de nós por segundo no PC mais atual, o que nos permite buscar aproximadamente 200 milhões de nós por movimento sob controles de tempo-padrão (três minutos por movimento). O fator de ramificação para xadrez é cerca de 35 em média, e 35^5 é aproximadamente igual a 50 milhões; assim, se usássemos a busca minimax, só poderíamos examinar cerca de cinco jogadas à frente. Embora não seja incompetente, tal programa pode ser enganado com facilidade por um jogador de xadrez humano médio, que ocasionalmente pode planejar seis ou oito jogadas à frente. Com a busca alfa-beta, chegamos a cerca de 10 jogadas, o que resulta em um nível de desempenho de especialista. A Seção 5.8 descreve técnicas adicionais de poda que podem estender a profundidade de busca efetiva a aproximadamente 14 jogadas. Para alcançar o *status* de grande mestre, precisaríamos de uma função de avaliação extensivamente ajustada e de um grande banco de dados de movimentos ótimos de abertura e de fim de jogo

5.4.4 Busca *versus* acesso

De alguma forma parece um exagero que um programa de xadrez inicie um jogo, considerando uma árvore de um bilhão de estados de jogo, apenas para concluir que moverá o seu peão para e4. Por cerca de um século, existem livros disponíveis sobre xadrez que descrevem boas jogadas na abertura e encerramento (Tattersall, 1911). Não é de estranhar, portanto, que muitos programas de jogo utilizem a *tabela de acesso* em vez de busca para abertura e encerramento dos jogos.

Para as aberturas, o computador conta principalmente com a perícia dos seres humanos. O melhor

conselho dos peritos humanos de como jogar cada abertura é copiado de livros e introduzido em tabelas para uso do computador. No entanto, os computadores também podem coletar estatísticas de um banco de dados de partidas previamente jogadas para ver que sequências de abertura conduzem a uma vitória na maioria das vezes. Nos movimentos iniciais há poucas opções e, desse modo, comentários de peritos e jogos passados dos quais extrair. Normalmente, depois de 10 movimentos termina-se em uma posição raramente vista, e o programa deve mudar da tabela de acesso para busca.

Perto do final do jogo há novamente poucas posições possíveis e, assim, mais chance de fazer acesso. Mas aqui é o computador que tem a experiência: a análise de computador de finais de jogos vai muito além de qualquer coisa alcançada pelos seres humanos. Um ser humano pode dizer-lhe a estratégia geral para jogar um final do jogo rei e torre *versus* rei (RTR): reduzir a mobilidade do rei oposto, encurralando-o em um canto do tabuleiro, utilizando o seu rei para evitar que o adversário escape do aperto. Outros finais, como rei, bispo e cavalo *versus* rei (RBCR), são difíceis de dominar e não têm uma descrição sucinta da estratégia. Um computador, por outro lado, pode *resolver* completamente o fim do jogo, produzindo um **programa de ação**, que é um mapeamento de todos os estados possíveis para a melhor jogada nesse estado. Então podemos apenas procurar pela melhor jogada em vez de recalculá-la novamente. Qual será o tamanho da tabela de acesso RBCR? Acontece que há 462 maneiras como dois reis podem ser colocados no tabuleiro sem estar adjacentes. Depois de os reis serem colocados, haverá 62 quadrados vazios para o bispo, 61 para o cavalo e dois jogadores possíveis para o próximo movimento; portanto, haverá apenas $462 \times 62 \times 61 \times 2 =$ 3.494.568 posições possíveis. Algumas dessas são xeque-mates; marque-as como tal em uma tabela. Em seguida, faça uma busca minimax **retrógrada:** reverter as regras do xadrez para retroceder em vez de mover. Qualquer movimento das brancas que, não importa com qual movimento as pretas respondam, termina em uma posição marcada como vitória, devendo ser também uma vitória. Continue essa busca até que todas as 3.494.568 posições estejam resolvidas como vitória, perda ou empate e você terá uma tabela de acesso infalível para todos os finais de jogos RBCR.

Utilizando essa técnica e com grande esforço de truques de otimização, Ken Thompson (1986, 1996) e Lewis Stiller (1992, 1996) resolveram todos os finais de jogos de xadrez com até cinco peças e alguns com seis peças, tornando-os disponíveis na Internet. Stiller descobriu um caso em que existia um mate forçado, mas exigia 262 movimentos, o que causou alguma consternação porque as regras do xadrez exigem que haja uma captura ou um movimento de um peão dentro de 50 lances. Mais tarde, o trabalho de Marc Bourzutschky e Yakov Konoval (Bourzutschky, 2006) resolveu todos os finais de jogo de seis peças sem peão e alguns de sete peças, havendo um final de jogo RTCRTBC (rei, torre, cavalo, rei, torre, bispo, cavalo) que com o melhor jogo requer 517 movimentos até a captura, que então conduz a um mate.

Se pudéssemos estender o tabuleiro de xadrez de final de jogo de seis para 32 peças, as brancas saberiam no movimento de abertura se seria uma vitória, perda ou empate. Isso não aconteceu até agora para o xadrez, mas tem acontecido para damas, como é explicado na seção de notas históricas.

5.5 JOGOS ESTOCÁSTICOS

Na vida real, existem muitos eventos externos imprevisíveis que podem nos colocar em situações inesperadas. Muitos jogos refletem essa imprevisibilidade, incluindo um elemento aleatório, como o lançamento de dados. Nós os chamamos de **jogos estocásticos**. O gamão é um jogo típico que combina sorte e habilidade. Os dados são rolados no início da ação de cada jogador para determinar os movimentos válidos. Por exemplo, na posição do jogo de gamão representada na Figura 5.10, as peças brancas tiveram uma rolagem de dados com seis e cinco pontos e têm quatro movimentos possíveis.

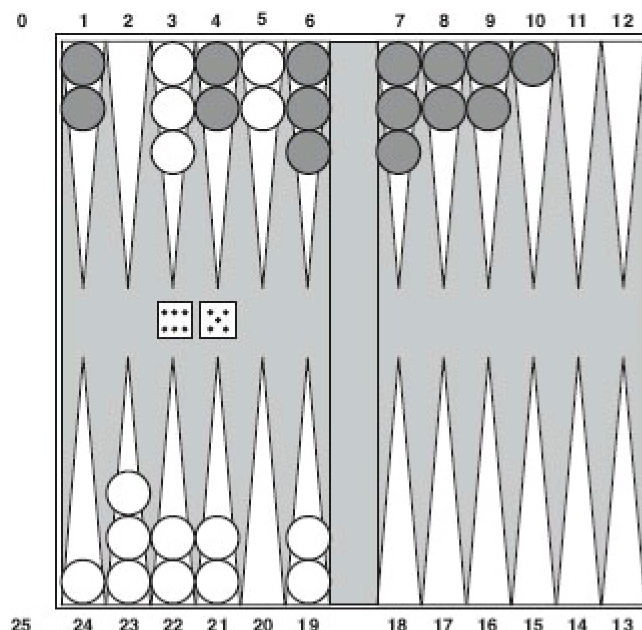


Figura 5.10 Uma posição típica em gamão. O objetivo do jogo é mover todas as peças para fora do tabuleiro. As brancas se movimentam no sentido horário (para a direita) até a posição 25, e as pretas se movimentam no sentido anti-horário (para a esquerda) até 0. Uma peça pode se mover para qualquer posição, a menos que existam várias peças oponentes nessa posição; se houver um oponente, ele será capturado e terá de recomeçar. Na posição mostrada, as brancas obtiveram 6 e 5 nos dados e devem escolher entre quatro movimentos válidos: (5–10, 5–11), (5–11, 19–24), (5–10, 10–16) e (5–11, 11–16), onde a notação (5–11, 11–16) significa mover uma peça da posição 5 para a 11 e depois mover uma peça da 11 para a 16.

Embora o jogador com as brancas saiba quais são seus próprios movimentos válidos, ele não sabe qual será a jogada das pretas e, portanto, não sabe quais serão os movimentos válidos das pretas. Isso significa que o jogador com as brancas não pode construir uma árvore de jogo-padrão do tipo que vimos em xadrez e no jogo da velha. Uma árvore de jogo em gamão deve incluir **nós de acaso** além de nós MAX e MIN. Os nós de acaso são mostrados como circunferências na Figura 5.11. As ramificações que levam a cada nó de acaso denotam as jogadas de dados possíveis, e cada uma é identificada com a jogada e a chance de que ela ocorra. Existem 36 maneiras de rolar dois dados, todas igualmente prováveis; porém, como 6–5 é igual a 5–6, existem apenas 21 lançamentos distintos. Os seis duplos (1–1 a 6–6) têm uma chance de 1/36; dizemos então que $P(1-1) = 1/36$. Os outros 15 lançamentos distintos têm a probabilidade de 1/18 cada.

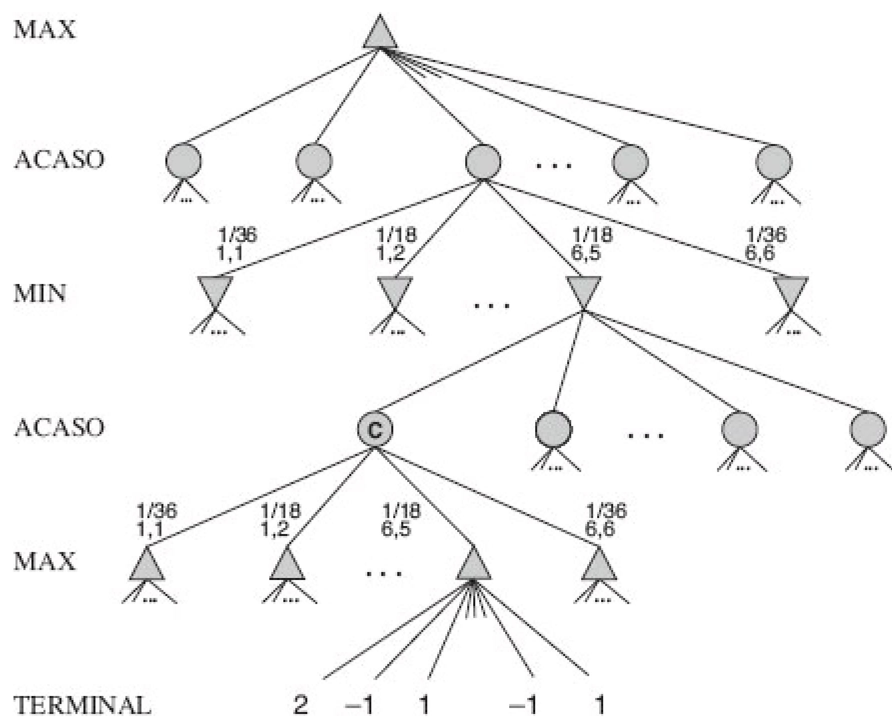


Figura 5.11 Árvore de jogo esquemática para uma posição de gamão.

A próxima etapa é entender como tomar decisões corretas. É óbvio que desejaremos escolher o movimento que leve à melhor posição. Porém, as posições resultantes não têm valores minimax definidos. Em vez disso, só podemos calcular o **valor esperado** de uma posição: a média sobre todos os resultados possíveis dos nós de acaso.

Isso nos leva a generalizar o **valor minimax** para jogos determinísticos até um **valor de expectiminimax** para jogos com nós de acaso. Nós terminais e nós de MAX e MIN (para os quais o lançamento de dados é conhecido) funcionam exatamente do mesmo modo que antes.

Para os nós de acaso calculamos o valor esperado, que é a soma do valor de todos os resultados, ponderada pela probabilidade de cada ação do acaso:

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILIDADE}(s) & \text{se TESTE DE TÉRMINO}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULTADO}(s, a)) & \text{se JOGADOR}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULTADO}(s, a)) & \text{se JOGADOR}(s) = \text{MIN} \\ \sum_r P(r)\, \text{EXPECTIMINIMAX}(\text{RESULTADO}(s, r)) & \text{se JOGADOR}(s) = \text{ACASO} \end{cases}$$

onde r representa um possível lançamento de dados (ou outro evento ao acaso) e RESULTADO (s, r) é o mesmo estado que s, com o fato adicional de que o resultado do lançamento de dados é r.

5.5.1 Funções de avaliação para os jogos de azar

Como ocorre no caso de minimax, a aproximação óbvia a fazer com expectiminimax é cortar a busca em certo ponto e aplicar uma função de avaliação a cada folha. Poderíamos pensar que as funções de avaliação de jogos como gamão devem ser exatamente como as funções de avaliação para xadrez — elas só precisam fornecer pontuações mais altas para posições melhores. Porém, de fato, a presença de nós de acaso significa que temos de ser mais cuidadosos sobre o significado dos valores

de avaliação. A Figura 5.12 mostra o que acontece: com uma função de avaliação que atribui valores [1, 2, 3, 4] às folhas, o movimento a_1 é melhor; com valores [1, 20, 30, 400], o movimento a_2 é melhor. Consequentemente, o programa se comportará de forma bastante diferente se fizermos uma mudança na escala de alguns valores de avaliação! Ocorre que, para evitar essa sensibilidade, a função de avaliação deve ser uma transformação linear positiva da probabilidade de vencer a partir de uma posição (ou, de modo mais geral, da utilidade esperada da posição). Essa é uma propriedade importante e geral de situações em que a incerteza está envolvida, como veremos com mais detalhes no Capítulo 16.

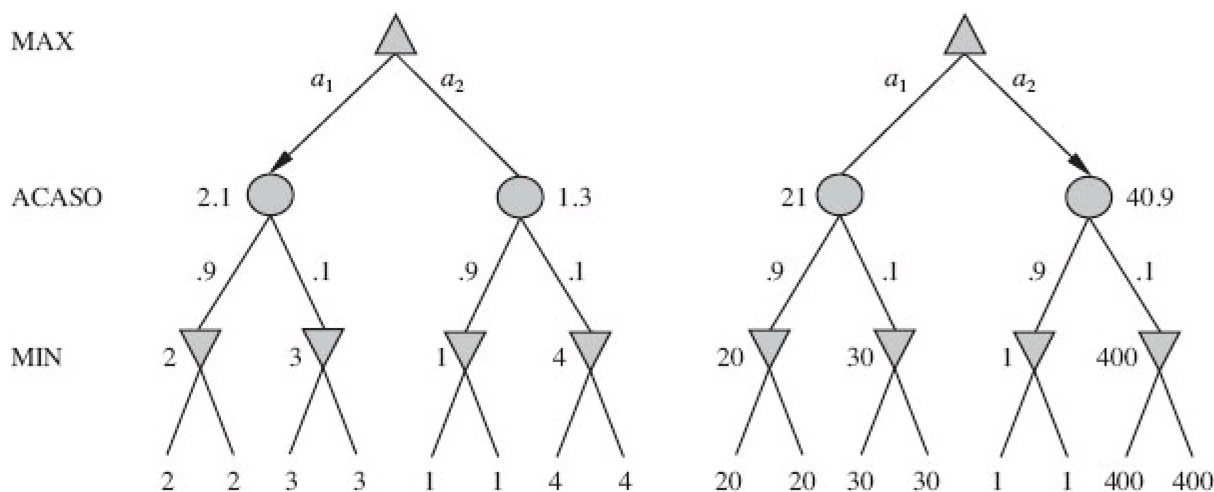


Figura 5.12 Uma transformação com preservação da ordem em valores de folhas altera o melhor movimento.

Se o programa conhecesse com antecedência todos os lançamentos de dados que ocorreriam no restante do jogo, a resolução de um jogo com dados seria muito semelhante à resolução de um jogo sem dados, o que minimax faz no tempo $O(b^m)$, onde b é o fator de ramificação e m é a profundidade máxima da árvore de jogo. Como expectiminimax também está considerando todas as sequências de lançamentos de dados possíveis, ele levará o tempo $O(b^m n^m)$, onde n é o número de lançamentos distintos.

Ainda que a profundidade da busca se limitasse a alguma profundidade pequena d, o custo extra comparado com o de minimax tornaria pouco realista considerar a possibilidade de examinar uma distância muito grande à frente na maioria dos jogos de azar. Em gamão, n é 21e b em geral é cerca de 20, mas, em algumas situações, ele pode chegar a 4.000 em lançamentos de dados que resultam em valores duplos. Talvez essas jogadas sejam tudo o que poderíamos administrar.

Outro modo de pensar no problema é: a vantagem de alfa-beta é que ela ignora desenvolvimentos futuros que simplesmente não irão acontecer, dada a melhor jogada. Desse modo, ela se concentra em ocorrências prováveis. Em jogos com dados, não há *nenhuma* sequência provável de movimentos porque, para que esses movimentos ocorressem, os dados primeiro teriam de cair da maneira correta para torná-los válidos. Esse é um problema geral sempre que a incerteza entra em cena: as possibilidades são enormemente multiplicadas, e a formação de planos de ação detalhados se torna inútil porque o mundo talvez não acompanhe o jogo.

Sem dúvida deve ter ocorrido ao leitor que talvez algo como a poda alfa-beta poderia ser aplicada a árvores de jogos com nós de acaso. Na verdade, isso é possível. A análise para nós de MIN e

MAX fica inalterada, mas também podemos podar nós de acaso, usando um pouco de engenhosidade. Considere o nó de acaso *C* da Figura 5.11 e o que acontece ao seu valor à medida que examinamos e avaliamos seus filhos. É possível encontrar um limite superior sobre o valor de *C* antes de examinarmos todos os seus filhos? (Lembre-se de que alfa-beta precisa disso para podar um nó e sua subárvore.) À primeira vista, pode parecer impossível porque o valor de *C* é a *média* dos valores de seus filhos e, a fim de calcular a média de um conjunto de números, temos que verificar todos os números. No entanto, se impusermos limites sobre os valores possíveis da função utilidade, poderemos chegar a limites para a média sem verificar todos os números. Por exemplo, se dissermos que todos os valores de utilidade estão entre –2 e +2, o valor de nós folhas será limitado e, nesse caso, *poderemos* impor um limite superior sobre o valor de um nó de acaso sem examinar todos os seus filhos.

Uma alternativa é fazer a **simulação de Monte Carlo** para avaliar uma posição. Comece com um algoritmo de busca alfa-beta (ou outro).

A partir de uma posição inicial, faça com que o algoritmo jogue milhares de jogos contra si mesmo, usando arremessos de dados aleatórios. No caso do gamão, o percentual de vitórias resultante tem se mostrado uma boa aproximação do valor da posição, mesmo que o algoritmo tenha uma heurística imperfeita e esteja em busca apenas de algumas jogadas (Tesauro, 1995). Para jogos com dados, esse tipo de simulação chama-se **lançamento.**

5.6 JOGOS PARCIALMENTE OBSERVÁVEIS

Muitas vezes, o xadrez tem sido descrito como guerra em miniatura, mas carece de pelo menos uma característica importante de guerras reais, chamada **observabilidade parcial**. Na “névoa da guerra”, a existência e a disposição das unidades inimigas é muitas vezes desconhecida até ser revelada por contato direto. Como resultado, a guerra inclui o uso de observadores e espiões para colher informações e uso de dissimulação e blefe para confundir o inimigo. Os jogos parcialmente observáveis compartilham essas características e são, portanto, qualitativamente diferentes dos jogos descritos nas seções anteriores.

5.6.1 Kriegspiel: xadrez parcialmente observável

Em jogos *determinísticos* parcialmente observáveis, a incerteza sobre o estado do tabuleiro resulta inteiramente da falta de acesso às escolhas feitas pelo adversário. Essa categoria inclui os jogos infantis como batalha naval (em que cada jogador coloca os navios em locais escondidos do adversário e não se movem) e Stratego (no qual a localização das peças é conhecida, mas os tipos das peças permanecem ocultos). Vamos examinar o jogo de **Kriegspiel**, uma variante parcialmente observável de xadrez em que as partes podem se mover, mas são completamente invisíveis para os adversários.

As regras do Kriegspiel são as seguintes: brancas e pretas veem um tabuleiro que contém apenas suas próprias peças. Um árbitro, que pode ver todas as peças, julga o jogo e faz anúncios periódicos

que os dois jogadores escutam. Por sua vez, as brancas propõem ao árbitro qualquer movimento que seria legal se não houvesse peças pretas. Se o movimento de fato não for legal (por causa das peças pretas), o árbitro anuncia "ilegal". Nesse caso, as brancas podem manter os movimentos propostos até que seja encontrado um legal — e aprendem mais sobre a localização das peças pretas no processo. Uma vez que seja proposto um lance legal, o árbitro anuncia uma ou mais das seguintes opções: "Capture no quadrado *X*" se houver uma captura, e "Xeque em *D*" se o rei preto estiver em xeque, onde *D* é a direção do xeque e pode ser um dos "cavalos", "linha", "coluna", "diagonal longa" ou "diagonal curta" (no caso de xeque a descoberto, o árbitro pode fazer dois anúncios de "xeque"). Se houver um xeque-mate ou afogamento nas pretas, o árbitro avisa; caso contrário, é a vez do lance do preto.

O Kriegspiel pode parecer terrivelmente impossível, mas os humanos o administram muito bem e os programas de computador estão começando a alcançá-lo. Ajuda lembrar a noção de **estado de crença**, como definido na Seção 4.4 e ilustrado na Figura 4.14 — o conjunto de todos os estados do tabuleiro *logicamente possível* dá o histórico completo das percepções até o momento. Inicialmente, o estado de crença branco é uma peça avulsa porque as peças pretas ainda não se moveram. Após um movimento das brancas e uma resposta das pretas, o estado de crença das brancas contém 20 posições porque as pretas têm 20 respostas a qualquer movimento das brancas. Manter o controle do estado de crença no decorrer do jogo é exatamente o problema da **estimativa de estado**, para o qual a etapa de atualização é dada na Equação 4.6. Podemos mapear a estimativa de estado do Kriegspiel diretamente para o quadro parcialmente observável, não determinístico da Seção 4.4, se considerarmos o adversário como fonte de não determinismo, ou seja, os RESULTADOS do movimento das brancas são compostos do resultado (previsível) do próprio movimento das brancas e pelo resultado imprevisível dado pela resposta das pretas.[3]

Dado um estado de crença atual, as brancas podem perguntar: "Posso ganhar o jogo?" Para um jogo parcialmente observável, a noção de **estratégia** é alterada; em vez de especificar um movimento a ser feito para cada *movimento* possível que o adversário possa fazer, precisamos de um movimento para cada *sequência de percepção* possível que possa ser recebida. Para o Kriegspiel, uma estratégia vencedora, ou **xeque-mate garantido**, é aquela que, para cada sequência de percepção possível, leva a um xeque-mate real para cada estado do tabuleiro possível no estado de crença atual, independentemente da forma como o adversário se move. Com essa definição, o estado de crença do adversário é irrelevante — a estratégia tem que funcionar, mesmo que o adversário possa ver todas as peças. Isso simplifica muito cálculo. A Figura 5.13 mostra parte de um xeque-mate garantido para um final de jogo RTR (rei, torre contra rei). Nesse caso, as pretas têm apenas uma peça (o rei); assim, um estado de crença para as brancas pode ser mostrado em um tabuleiro simples marcando cada posição possível do rei preto.

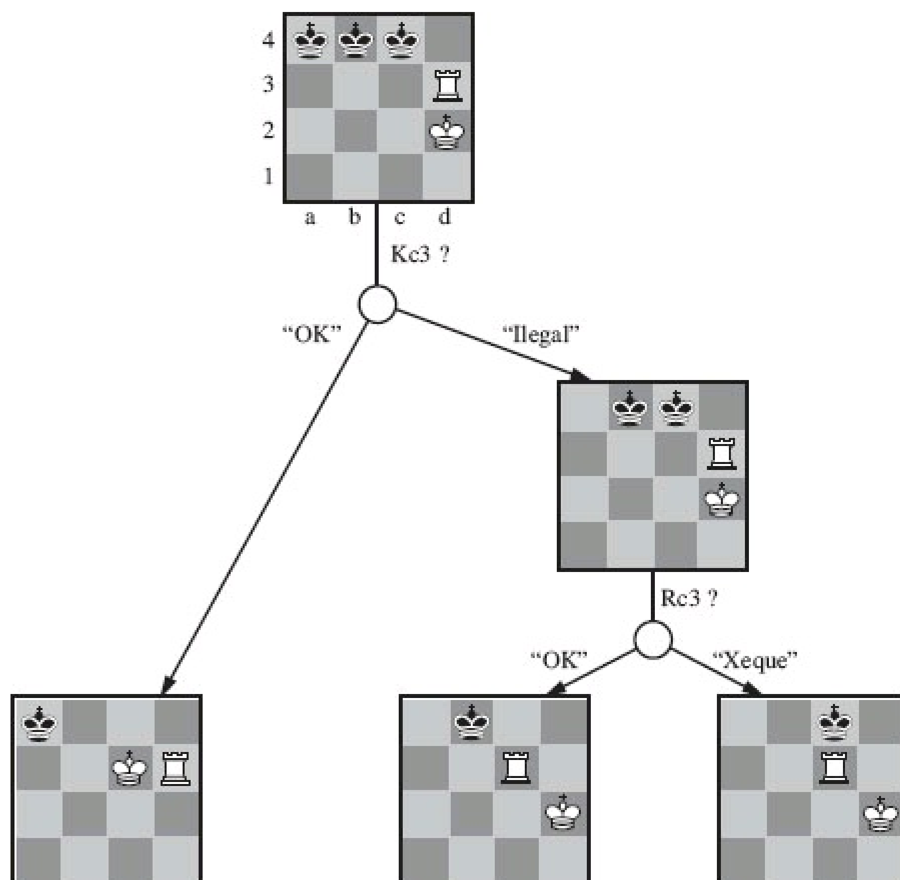


Figura 5.13 Parte de um xeque-mate garantido RTR, mostrado ao final do jogo, em um tabuleiro reduzido. No estado de crença inicial, o rei preto está em uma das três localizações possíveis. Através de uma combinação de movimentos de sondagem, a estratégia se reduz para um. A conclusão do xeque-mate é deixada como exercício.

O algoritmo genérico de busca E-OU pode ser aplicado no espaço de estado de crença para encontrar os xeque-mates garantidos, como na Seção 4.4. O algoritmo de estado de crença incremental mencionado na seção muitas vezes encontra xeque-mates no meio do jogo até uma profundidade 9 — provavelmente muito além das habilidades dos jogadores humanos.

Além dos xeque-mates garantidos, o Kriegspiel admite um conceito inteiramente novo que não faz sentido em jogos totalmente observáveis: **xeque-mate probabilístico**. Ainda é requerido que tais xeque-mates funcionem em cada estado do tabuleiro em estado de crença; são probabilísticos com respeito à randomização dos movimentos do jogador vencedor. Para obter a ideia básica, considere o problema de encontrar um rei preto solitário utilizando apenas o rei branco. Simplesmente movendo de forma aleatória, o rei branco *eventualmente* dará de cara com o rei preto, mesmo que este último tente evitar esse destino, desde que o preto não pode se manter imaginando movimentos evasivos corretos indefinidamente. Na terminologia da teoria da probabilidade, a detecção ocorre *com probabilidade* 1. O final de jogo RBCR — rei, bispo e cavalo, contra o rei — está ganha nesse sentido; a branca apresenta à preta uma sequência infinita aleatória de escolhas, para uma das quais a preta vai imaginar incorretamente e revelar sua posição, levando ao xeque-mate. O fim do jogo RBBR, por outro lado, se ganha com probabilidade $1 - \epsilon$.

A branca pode forçar uma vitória apenas, deixando um de seus bispos sem proteção em um movimento. Se acontecer de a preta estar no lugar certo e capturar o bispo (um movimento que iria perder se o bispo estivesse protegido), a partida estará empatada. A branca pode optar por um

movimento arriscado em algum ponto escolhido randomicamente no meio de uma distância muito longa, reduzindo assim ∊ para uma constante arbitrariamente pequena, mas não pode reduzir ∊ para zero.

É muito raro que um xeque-mate garantido ou probabilístico possa ser encontrado em qualquer profundidade razoável, exceto ao final do jogo. Às vezes, uma estratégia de xeque-mate funciona para *alguns* dos estados do tabuleiro no estado de crença atual, mas não para outros. Tentar tal estratégia pode ter sucesso, levando a um **xeque-mate acidental** — acidental no sentido de que a branca não poderia *saber* que seria xeque-mate — se acontecer de as peças pretas estarem nos lugares certos (a maioria dos xeque-mates em jogos entre os seres humanos é dessa natureza acidental). Essa ideia leva naturalmente à questão de *o quanto provável* é que determinada estratégia irá vencer, o que leva, por sua vez, à questão do *quanto é provável* que cada estado do tabuleiro, no estado de crença atual, é o verdadeiro estado do tabuleiro.

Uma primeira inclinação deverá ser propor que todas as posições do tabuleiro, no estado de crença atual, sejam igualmente prováveis, mas isso pode não estar certo. Considere, por exemplo, o estado de crença da branca após o primeiro lance do jogo da preta. Por definição (assumindo que a preta desempenha otimamente), a preta deve ter jogado um ótimo lance, e assim deveria ser atribuída uma probabilidade zero a todas as posições do tabuleiro resultantes de lances subótimos. Esse argumento não é muito certo também porque o objetivo de *cada jogador não é apenas mover as peças para os quadrados à direita, mas também minimizar a informação que o adversário tem sobre sua localização.* Jogar qualquer estratégia "ótima" previsível fornece informações ao adversário. Por isso, o jogo ideal em jogos parcialmente observáveis requer uma vontade de jogar de alguma forma *ao acaso* (é por isso que os inspetores de higiene de restaurante fazem visitas de inspeção aleatórias). Isso significa selecionar ocasionalmente lances que podem parecer "intrinsecamente" fracos, mas ganham força a partir de sua forte imprevisibilidade porque é improvável que o adversário tenha preparado qualquer defesa contra eles.

A partir dessas considerações, parece que as probabilidades associadas com as posições do tabuleiro no estado de crença atual só podem ser calculadas a partir de uma estratégia ótima randomizada; por sua vez, o cálculo dessa estratégia parece exigir conhecimento das probabilidades das diversas posições que o tabuleiro possa ter. Esse enigma pode ser resolvido adotando a noção da teoria dos jogos de uma solução de **equilíbrio**, que será abrangida no Capítulo 17. Um equilíbrio especifica uma estratégia ótima randomizada para cada jogador. Porém, calcular o equilíbrio é proibitivamente caro, mesmo para pequenos jogos, e está fora de questão para o Kriegspiel. Atualmente, o projeto de algoritmos eficientes para o jogo geral de Kriegspiel é um tópico de busca aberta. A maioria dos sistemas realiza a perspectiva de profundidade limitada em seu próprio espaço de estado de crença, ignorando o estado de crença do adversário. As funções de avaliação são semelhantes às do jogo observável, mas incluem um componente para o tamanho do estado de crença — quanto menor, melhor!

5.6.2 Jogos de cartas

Os jogos de cartas dão muitos exemplos de observabilidade parcial *estocástica*, onde a falta de

informação é gerada aleatoriamente. Por exemplo, em muitos jogos, no início do jogo as cartas são distribuídas aleatoriamente, com cada jogador recebendo uma mão que não é visível para os outros jogadores. Tais jogos incluem *bridge*, *whist*, *hearts* e algumas formas de pôquer.

À primeira vista, pode parecer que esses jogos de cartas são como jogos de dados: as cartas são distribuídas de forma aleatória e determinam as jogadas disponíveis para cada jogador, mas no início todos os "dados" são jogados! Mesmo que essa analogia acabe sendo incorreta, ela sugere um algoritmo efetivo: considerar todas as distribuições possíveis das cartas invisíveis, resolver cada uma como se fosse um jogo totalmente observável, e então escolher o lance que tem a melhor média de resultado sobre todos os lances. Suponha que cada mão ocorra com a probabilidade $P(s)$, então o lance que queremos é

$$\operatorname*{argmax}_{a} \sum_{s} P(s) \text{ MINIMAX (RESULTADO } (s, a)). \tag{5.1}$$

Aqui, executaremos o MINIMAX exato se computacionalmente viável; caso contrário, executaremos o H-MINIMAX.

Agora, na maioria dos jogos de carta, o número de mãos possíveis é bastante grande. Por exemplo, no jogo de *bridge*, cada jogador vê apenas duas das quatro mãos; existem duas mãos invisíveis, de 13 cartas cada, então o número de mãos é (26/13) = 10.400.600. A resolução de uma mão é muito difícil, por isso resolver 10 milhões está fora de questão. Em vez disso, recorreremos a uma aproximação de Monte Carlo: em vez de somar *todas* as rodadas, tomamos uma *amostra aleatória* de N rodadas, onde a probabilidade de a rodada s aparecer na amostra é proporcional a $P(s)$:

$$\operatorname*{argmax}_{a} \frac{1}{N} \sum_{i=1}^{N} \text{MINIMAX}(\text{RESULT}(s_i, a)) \; . \tag{5.2}$$

[Observe que $P(s)$ não aparece explicitamente no somatório porque as amostras já estão extraídas de acordo com $P(s)$.] À medida que N aumenta, a soma sobre a amostra aleatória tende ao valor exato, mas mesmo para N relativamente pequeno — digamos, de 100 a 1.000 — o método dá uma boa aproximação. Pode também ser aplicado a jogos determinísticos, como o de Kriegspiel, a partir de algumas estimativas razoáveis de $P(s)$.

Para jogos como *whist* e *hearts*, nos quais não há fase de lance ou de apostas antes de o jogo começar, cada rodada será igualmente provável e, assim, os valores de $P(s)$ são todos iguais. Para o *bridge*, o jogo é precedido por uma fase de leilão, em que cada equipe indica quantos *tricks* espera ganhar. Como os jogadores fazem seus lances com base em suas cartas, os outros jogadores aprendem mais sobre a probabilidade de cada lance. Levar isso em conta para decidir como jogar a mão é complicado, pelas razões mencionadas em nossa descrição do Kriegspiel: os jogadores podem fazer os lances de forma a minimizar a informação transmitida aos seus adversários. Mesmo assim, a abordagem é bastante eficaz para o *bridge*, como mostraremos na Seção 5.7.

A estratégia descrita nas Equações 5.1 e 5.2 chama-se às vezes *média sobre clarividência* porque assume que o jogo vai se tornar observável para ambos os jogadores imediatamente após o primeiro lance. Apesar de seu apelo intuitivo, a estratégia pode conduzir a um extravio. Considere a seguinte história:

Dia 1: A estrada *A* leva a um monte de moedas de ouro; a estrada *B* leva a uma bifurcação. Tome a estrada da esquerda e você encontrará uma grande pilha de ouro; tome a estrada da direita e você será atropelado por um ônibus.

Dia 2: A estrada *A* leva a um monte de moedas de ouro; a estrada *B* leva a uma bifurcação. Tome a estrada da direita e você encontrará uma grande pilha de ouro; tome a estrada da esquerda e você será atropelado por um ônibus.

Dia 3: A estrada *A* leva a um monte de moedas de ouro; a estrada *B* leva a uma bifurcação. Um ramo da bifurcação leva a uma grande pilha de ouro, mas, se tomar o caminho errado será atropelado por um ônibus. Infelizmente, você não sabe qual é o correto.

A média sobre clarividência leva ao seguinte raciocínio: no dia 1, *B* é a escolha certa; no dia 2, *B* é a escolha certa; no dia 3, a situação é a mesma que a do dia 1 ou 2, então *B* deverá ser ainda a escolha certa.

Agora podemos ver como uma média sobre a clarividência falha: não considera o *estado de crença* em que o agente estará depois da ação. Um estado de crença de total ignorância não é desejável, especialmente quando uma possibilidade certa é a morte. Por assumir que cada estado futuro será automaticamente de perfeito conhecimento, a abordagem nunca seleciona ações que *reúnem informações* (como o primeiro lance na Figura 5.13); nem vai escolher as ações que escondem informação do adversário ou fornece informação a um parceiro porque se assume que eles já conhecem as informações e ela nunca vai **blefar** no pôquer,[4] pois assume que o adversário pode ver as suas cartas. No Capítulo 17, vamos mostrar como construir algoritmos que fazem todas essas coisas pela virtude de resolver o verdadeiro problema de decisão parcialmente observável.

5.7 PROGRAMAS DE JOGOS DE ÚLTIMA GERAÇÃO

Em 1965, o matemático russo Alexander Kronrod chamou o xadrez de “*Drosophila* inteligência artificial”. John McCarthy discorda: enquanto os geneticistas usam moscas de frutas para fazer descobertas que se aplicam à biologia de forma mais ampla, a IA usou o xadrez para fazer o equivalente à criação de moscas de fruta muito rápidas. Talvez a melhor analogia seja de que o xadrez é para a IA o mesmo que a corrida automobilística de Grand Prix é para a indústria do automóvel: programas de última geração de jogo são incrivelmente rápidos, máquinas altamente otimizadas, que incorporam os últimos avanços da engenharia, mas eles não são muito úteis para fazer as compras ou dirigir fora da estrada. No entanto, corridas e jogos geram excitação e um fluxo constante de inovações que são adotadas por uma comunidade maior. Nesta seção, verificaremos o que é preciso para se sair bem em vários jogos.

Xadrez: o programa de xadrez da IBM, Deep Blue, agora aposentado, foi muito conhecido por derrotar o campeão mundial Garry Kasparov em um jogo de exibição amplamente divulgado. O Deep Blue executou em um computador paralelo com 30 processadores IBM RS/6000 fazendo busca alfa-beta. A única parte era uma configuração personalizada de 480 processadores de xadrez VLSI que realizava a geração de movimento e de ordenação para os últimos poucos níveis da árvore e avaliava os nós folha. O Deep Blue buscava até 30 bilhões de posições por movimento, alcançando

rotineiramente uma profundidade igual a 14. O coração da máquina é uma busca alfa-beta de aprofundamento iterativo padrão com uma tabela de transposição, mas a chave de seu sucesso parece ter sido sua habilidade de gerar extensões além do limite de profundidade para linhas suficientemente interessantes de movimentos forçados. Em alguns casos, a busca alcançou uma profundidade de 40 jogadas. A função de avaliação tinha mais de 8.000 características, muitas delas descrevendo padrões de peças altamente específicos. Foi usado um "livro de aberturas" com aproximadamente 4.000 posições, bem como um banco de dados de 700.000 jogos de grandes mestres a partir do qual podiam ser extraídas recomendações consensuais. O sistema também utilizava um grande banco de dados de finais de jogos com posições resolvidas, contendo todas as posições com cinco peças e muitas com seis peças. Esse banco de dados tem o efeito de estender de forma significativa a profundidade efetiva da busca, permitindo ao Deep Blue jogar com perfeição em alguns casos, até mesmo quando está a muitos movimentos de distância do xeque-mate.

O sucesso do Deep Blue reforçou a ampla convicção de que o progresso dos jogos de computadores vinha principalmente de um hardware cada vez mais poderoso — uma visão encorajada pela IBM. Mas as melhorias algorítmicas têm permitido aos programas executar em PCs-padrão, para ganhar campeonatos mundiais de xadrez por computador. Diversas heurísticas de poda são usadas para reduzir o fator de ramificação efetivo a menos de 3 (comparado ao fator de ramificação real de aproximadamente 35). A mais importante delas é a heurística de **movimento nulo**, que gera um bom limite inferior sobre o valor de uma posição, usando uma busca rasa na qual o oponente chega ao dobro de movimentos no início. Esse limite inferior frequentemente permite a poda alfa-beta sem o custo de uma busca em profundidade total. Também é importante a **poda de futilidades**, que ajuda a decidir com antecedência que movimentos causarão um corte beta nos nós sucessores.

O HYDRA pode ser visto como o sucessor do Deep Blue. O HYDRA executa em um cluster de processador de 64 com 1 gigabyte por processador e com hardware personalizado em forma de chips FPGA (*field programmable gate array*: arranjo de portas programável em campo). O HYDRA atinge 200 milhões de avaliações por segundo, o mesmo que o Deep Blue, mas alcança 18 camadas de profundidade, em vez de apenas 14, por causa do uso agressivo da heurística de movimento nulo e poda adiantada.

O RYBKA, vencedor do Campeonato Mundial de Xadrez de Computador de 2008 e 2009, é considerado o computador jogador mais forte do momento. Ele utiliza um processador Xeon de 8-core e 3,2 GHz Intel imediatamente disponível, mas pouco se sabe sobre a concepção do programa. Sua principal vantagem parece ser a sua função de avaliação, que foi ajustada pelo seu desenvolvedor principal, o mestre internacional Vasik Rajlich, e pelo menos três outros grandes mestres.

Os jogos mais recentes sugerem que os programas de computador mais avançados de xadrez impulsionaram todos os competidores humanos (veja as observações históricas para detalhes).

Jogo de damas: Jonathan Schaeffer e seus colegas desenvolveram o CHINOOK, que executa em PCs e utiliza busca alfa-beta. O Chinook derrotou o campeão humano de longa data em uma partida abreviada em 1990, e desde 2007 tem sido capaz de jogar perfeitamente utilizando busca alfa-beta combinada com uma base de dados de 39 trilhões de posições finais.

O jogo **Othello**, também chamado Reversi, provavelmente é mais popular como jogo de computador do que como jogo de tabuleiro. Ele tem um espaço de busca menor que o do xadrez, em geral de 5-15 movimentos válidos, mas a experiência de avaliação teve de ser desenvolvida desde o início. Em 1997, o programa Logistello (Buro, 2002) derrotou o campeão mundial humano, Takeshi Murakami, por seis jogos a zero. De modo geral, todos reconhecem que os seres humanos não são capazes de superar os computadores no Othello.

Gamão: A Seção 6.5 explicou por que a inclusão da incerteza dos lançamentos de dados torna uma busca profunda um luxo dispendioso. A maior parte do trabalho em gamão se dedicou a melhorar a função de avaliação. Gerry Tesauro (1992) combinou o método de aprendizado de reforço de Samuel com as técnicas de redes neurais para desenvolver um avaliador notavelmente preciso que é utilizado com uma busca até a profundidade 2 ou 3. Depois de disputar mais de um milhão de jogos de treinamento contra si mesmo, o programa de Tesauro, TD-GAMMON ficou competitivo com os melhores jogadores humanos. Em alguns casos, as opiniões do programa nos movimentos de abertura do jogo alteraram de forma radical a sabedoria adquirida.

O **Go** é o jogo de tabuleiro mais popular na Ásia, exigindo de seus profissionais, no mínimo, tanta disciplina quanto o xadrez. Como o tabuleiro tem 19 × 19 e os lances são permitidos na entrada de (quase) todos os quadrados em branco, o fator de ramificação começa em 361, um valor assustador para os métodos de busca alfa-beta comum. Além disso, é difícil escrever uma função de avaliação porque o controle do território é muitas vezes imprevisível até o fim do jogo. Portanto, os programas de topo, como o MoGo, evitam a busca alfa-beta e, em vez disso, utilizam os lançamentos de Monte Carlo. A malícia é decidir que lances fazer no curso do lançamento. Não há poda agressiva; todos os movimentos são possíveis. O método de limites de confiança superior em árvores funciona ao fazer movimentos aleatórios nas primeiras poucas iterações e, ao longo do tempo, guiando o processo de amostragem para selecionar os lances que levaram a vitórias nas amostras anteriores. São adicionados alguns truques, incluindo *regras baseadas em conhecimento* que sugerem lances em particular sempre que determinado padrão foi detectado e *limitou a busca local* para decidir questões táticas. Alguns programas também incluem técnicas especiais da **teoria combinatória dos jogos** para analisar finais de jogos. Essas técnicas decompõem uma posição em suposições que podem ser analisadas separadamente e depois combinadas (Berlekamp e Wolfe, 1994; Muller, 2003). As soluções ótimas obtidas dessa forma têm surpreendido muitos jogadores de Go profissional, que pensavam que estavam jogando otimamente o tempo todo. Os programas atuais para Go funcionam em nível proprietário em tabuleiro 9 × 9 reduzido, mas ainda estão em nível amador avançado em um tabuleiro completo.

O ***bridge*** é um jogo de cartas de informações imperfeitas: as cartas de um jogador ficam ocultas dos outros jogadores. O *bridge* também é um jogo de *vários participantes* com quatro jogadores em vez de dois, embora os jogadores formem duas equipes. Como vimos na Seção 5.6, o jogo ótimo em jogos parcialmente observáveis como o *bridge* pode incluir elementos de aquisição de informações, comunicação, blefe e cuidadosa ponderação de probabilidades. Muitas dessas técnicas são usadas no programa Bridge Baron™ (Smith *et al*., 1998), que venceu o campeonato de *bridge* por computador de 1997. Embora não jogue muito bem, o Bridge Baron é um dos poucos sistemas de jogos bem-sucedidos a usar planos hierárquicos complexos (veja o Capítulo 11) que envolvem ideias de alto

nível como **trapacear** e **forçar o descarte de trunfo**, familiares aos jogadores de *bridge*.

O programa GIB (Ginsberg, 1999) venceu o campeonato de *bridge* de 2000 decisivamente usando o método de Monte Carlo. Desde então, outros programas vencedores seguiram a conduta do GIB. A maior inovação do GIB é utilizar **generalização baseada em explanação** para calcular e armazenar na cache regras gerais para desempenho ótimo em várias classes-padrão de situações, em vez de avaliar cada situação individualmente. Por exemplo, em uma situação em que um jogador tem as cartas A-K-Q-J-4-3-2 de um naipe e outro jogador tem 10-9-8-7-6-5, existem 7 × 6 = 42 maneiras para o primeiro jogador encabeçar e o segundo jogador seguir. Mas o GIB trata essas situações apenas como duas: o primeiro jogador pode tirar uma carta alta ou uma baixa; as cartas exatas não importam. Com essa otimização (e algumas outras), o GIB pode resolver as 52 cartas, uma mão plenamente observável *exatamente* em cerca de um segundo. A precisão tática do GIB contribui para sua inabilidade de raciocinar sobre as informações. Ele terminou em 12º lugar em um grupo de 35 no concurso de pares (envolvendo apenas o jogo da mão, não os lances) no campeonato mundial de 1998 para jogadores humanos, superando de longe as expectativas de muitos especialistas.

Existem várias razões para o desempenho do GIB em nível especialista com a simulação de Monte Carlo, considerando que os programas do Kriegspiel não o fazem. Primeiro, a avaliação de GIB da versão totalmente observável do jogo é exata, buscando da árvore de jogo completa, enquanto os programas Kriegspiel dependem da heurística inexata. Mas muito mais importante é o fato de que, no *bridge*, a maior parte da incerteza nas informações parcialmente observáveis vem da aleatoriedade do lance, não do jogo do adversário. A simulação de Monte Carlo trata bem a aleatoriedade, mas nem sempre lida bem com a estratégia, especialmente quando a estratégia envolve o valor da informação.

Palavras cruzadas: A maioria das pessoas acha que a parte mais difícil das palavras cruzadas é vir à baila boas palavras, mas, dado o dicionário oficial, acaba por ser bastante fácil programar um gerador de lance para encontrar o lance de pontuação mais alta (Gordon, 1994). No entanto, isso não significa que o jogo seja resolvido: tomar apenas o lance de melhor pontuação a cada vez é resultado de um jogador bom, mas não especialista. O problema é que a palavra cruzada é ao mesmo tempo parcialmente observável e estocástica: você não sabe que letras o outro jogador tem ou quais as próximas letras que você vai puxar. Portanto, jogar palavras cruzadas bem combina as dificuldades do gamão com o *bridge*. No entanto, em 2006, o programa QUACKLE derrotou o ex-campeão mundial, David Boys, por 3×2.

5.8 ABORDAGENS ALTERNATIVAS

Tendo em vista que o cálculo de decisões ótimas em jogos é intratável na maioria dos casos, todos os algoritmos devem fazer algumas suposições e aproximações. A abordagem-padrão, baseada em minimax, funções de avaliação e alfa-beta, é apenas uma maneira de fazer isso. Talvez porque tenha funcionado por tanto tempo, a abordagem-padrão foi desenvolvida intensivamente e domina outros métodos em jogos de torneios. Alguns especialistas no campo acreditam que isso tenha feito os jogos se divorciarem da parte principal da busca de IA: porque a abordagem-padrão não oferece mais

tanto espaço para novas ideias sobre questões gerais de tomada de decisões. Nesta seção, veremos as alternativas.

Primeiro, vamos considerar a heurística minimax. Ela seleciona um movimento ótimo em dada árvore de busca *desde que as avaliações de nós de folhas sejam exatamente corretas*. Na realidade, as avaliações são normalmente estimativas brutas do valor de uma posição e podemos considerar que há grandes erros associados a elas. A Figura 5.14 mostra uma árvore de jogo de duas jogadas para a qual o minimax sugere tomar o ramo da direita, porque 100 > 99. Esse é o lance correto se as avaliações estiverem todas corretas. Mas é claro que a função de avaliação é apenas aproximada. Suponha que a avaliação de cada nó tenha um erro que é independente de outros nós e aleatoriamente distribuído com média zero e desvio padrão. Então, quando $\sigma = 5$, o ramo esquerdo é realmente melhor 71% do tempo e 58% do tempo quando $\sigma = 2$. A percepção por trás disso é que o ramo do lado direito tem quatro nós que estão próximos de 99; se um erro na avaliação de qualquer um dos quatro fizer com que o ramo da direita deslize abaixo de 99, então o ramo da esquerda é o melhor.

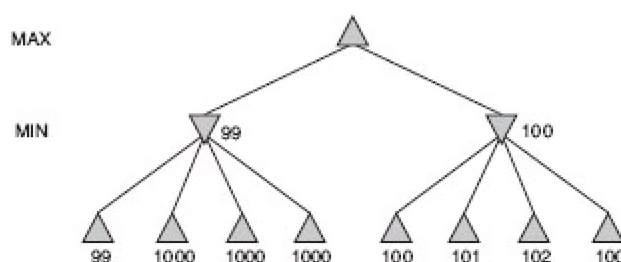


Figure 5.14 Uma árvore de jogo de duas jogadas para a qual o minimax pode ser inadequado.

Na realidade, as circunstâncias são realmente piores do que isso porque o erro na função de avaliação *não* é independente. Se obtivermos um nó errado, as chances são altas de que os próximos nós na árvore também estarão errados. Porém, o fato de o nó identificado com 99 ter irmãos identificados com 1.000 sugere que, de fato, ele pode ter um valor verdadeiro mais alto. Podemos usar uma função de avaliação que retorna uma *distribuição de probabilidades* sobre valores possíveis, mas é difícil combinar essas distribuições corretamente porque não vamos ter um bom modelo das dependências muito fortes que existem entre os valores dos nós irmãos.

Em seguida, consideramos o algoritmo de busca que gera a árvore. O objetivo do projetista de algoritmos é especificar uma computação que funcione com rapidez e que gere um bom movimento. Os algoritmos alfa–beta foram projetados não apenas para selecionar um bom movimento, mas também para calcular limites sobre os valores de todos os movimentos válidos. Para ver por que essas informações extras são desnecessárias, considere uma posição em que só existe um movimento válido. A busca alfa−-beta ainda irá gerar e avaliar uma árvore de busca grande, dizendo que o único lance é o melhor lance e atribuindo-lhe um valor. Mas, como temos que fazer o movimento de qualquer forma, conhecer o valor do movimento é inútil. Da mesma forma, se houver um lance obviamente bom e vários lances que são legais, mas levam a uma perda rápida, não vamos querer que o alfa-beta desperdice tempo determinando um valor preciso para apenas um bom lance. Melhor fazer apenas o movimento rapidamente e economizar tempo para mais tarde. Isso leva à ideia da *utilidade de uma expansão de nó*. Um bom algoritmo de busca deve selecionar expansões de nós de utilidade elevada, ou seja, aquelas que deverão levar à descoberta de um movimento significativamente melhor. Se não houver nenhuma expansão de nó cuja utilidade seja mais alta que

seu custo (em termos de tempo), o algoritmo deve interromper a busca e efetuar um movimento. Note que isso funciona não apenas para situações de claro favoritismo, mas também no caso de *movimentos simétricos*, para os quais nenhuma quantidade de busca mostrará que um movimento é melhor que outro.

Esse tipo de raciocínio que trata dos resultados obtidos com a computação é chamado **metarraciocínio** (raciocínio sobre o raciocínio). Ele se aplica não apenas aos jogos, mas a qualquer espécie de raciocínio. Todas as computações são feitas com a finalidade de tentar alcançar decisões melhores, todas têm custos e todas têm alguma probabilidade de resultar em certa melhoria na qualidade da decisão. A alfa-beta incorpora o tipo mais simples de metarraciocínio, ou seja, um teorema para o efeito de que certas ramificações da árvore podem ser ignoradas sem perda. É possível fazer muito melhor. No Capítulo 16, veremos como essas ideias podem se tornar exatas e implementáveis.

Finalmente, vamos reexaminar a natureza da própria busca. Os algoritmos para busca heurística e para jogos funcionam gerando sequências de estados concretos, começando pelo estado inicial e depois aplicando uma função de avaliação. É claro que não é assim que os seres humanos jogam. No xadrez, com frequência se tem em mente um objetivo específico — por exemplo, preparar uma armadilha para a rainha do oponente — e se pode usar esse objetivo para gerar *seletivamente* planos plausíveis para alcançá-lo. Esse tipo de raciocínio orientado para objetivos ou planejamento às vezes elimina por completo a busca combinatória. O Paradise de David Wilkins (1980) é o único programa a usar com sucesso o raciocínio orientado para objetivos no xadrez: ele foi capaz de resolver alguns problemas de xadrez que exigiam uma combinação de 18 movimentos. Até agora não existe nenhuma compreensão razoável de como *combinar* os dois tipos de algoritmos para formar um sistema eficiente e robusto, embora o Bridge Baron possa ser um passo na direção correta. Um sistema totalmente integrado seria uma realização significativa não apenas para a pesquisa na área de jogos, mas também para a pesquisa em IA em geral porque seria uma boa base para um agente inteligente geral.

5.9 RESUMO

Examinamos uma variedade de jogos para entender o que significa um jogo ótimo e para compreender como jogar bem na prática. As ideias mais importantes são:

- Um jogo pode ser definido pelo **estado inicial** (a forma como o tabuleiro é configurado), pelas **ações** válidas em cada estado, o **resultado** de cada ação, um **teste de término** (que informa quando o jogo é encerrado) e por uma **função utilidade** que se aplica a estados terminais.
- Em jogos de dois jogadores com soma zero e **informações perfeitas**, o algoritmo **minimax** pode selecionar movimentos ótimos usando uma enumeração da árvore de jogo em profundidade.
- O algoritmo de busca **alfa-beta** calcula o mesmo movimento ótimo que o minimax, mas alcança uma eficiência muito maior pela eliminação de subárvores comprovadamente irrelevantes.
- Em geral, não é possível considerar a árvore de jogo inteira (mesmo com alfa-beta) e, assim, precisamos cortar a busca em algum ponto e aplicar uma **função de avaliação** que fornece uma

estimativa da utilidade de um estado.

- Muitos programas de jogos pré-calculam tabelas das melhores jogadas no início e no final do jogo para que possam consultar uma jogada em vez de buscar.
- Os jogos de azar podem ser tratados por uma extensão do algoritmo minimax que avalia um **nó de acaso** tomando a utilidade média de todos os seus nós filhos, ponderada pela probabilidade de cada filho.
- O desempenho ótimo emjogos de **informações imperfeitas**, como o Kriegspiel e o *bridge*, exige raciocínio sobre os **estados de crença** corrente e futura de cada jogador. Uma aproximação simples pode ser obtida calculando-se a média dos valores de uma ação sobre cada configuração possível de informações omitidas.
- Os programas têm superado até mesmo jogadores humanos que são campeões em jogos, como xadrez, damas e Othello. Os seres humanos mantêm vantagem em vários jogos de informação imperfeita, como *bridge*, pôquer e Kriegspiel, e em jogos com fatores muito grandes de ramificação e pouco conhecimento heurístico bom, como o Go.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

A história inicial dos jogos mecânicos foi marcada por numerosas fraudes. A mais notória dessas fraudes foi a do barão Wolfgang von Kempelen (1734-1804, "o turco"), um suposto autômato jogador de xadrez que derrotou Napoleão antes de ser exposto como um armário de truques de mágica que alojava um especialista humano em xadrez (consulte Levitt, 2000). Ele jogou de 1769 até 1854. Em 1846, Charles Babbage (que tinha ficado fascinado com o turco) parece ter colaborado para a primeira discussão séria sobre a viabilidade dos jogos de xadrez e damas por computador (Morrison e Morrison, 1961). Ele não entendeu a complexidade exponencial das árvores de busca, alegando que "as combinações envolvidas na máquina analítica superavam enormemente qualquer uma requerida, até mesmo pelo jogo de xadrez". Babbage também projetou, mas não construiu, uma máquina de uso especial para jogar o jogo da velha. A primeira máquina para jogos verdadeira foi construída por volta de 1890 pelo engenheiro espanhol Leonardo Torres y Quevedo. Ele se especializou no final do jogode xadrez "KRK" (rei e torre contra rei), garantindo uma vitória com rei e torre a partir de qualquer posição.

Em geral, as origens do algoritmo minimax se localizam em um artigo publicado em 1912 por Ernst Zermelo, o desenvolvedor da moderna teoria de conjuntos. Infelizmente, o artigo continha vários erros e não descrevia o minimax de forma correta. Por outro lado, ele delineou as ideias da análise retrógrada e propôs (mas não provou) o que ficou conhecido como teorema de Zermelo: que o xadrez é determinado — as brancas ou as pretas podem forçar uma vitória ou é um empate, só não sabemos qual das opções. Zermelo diz que, se eventualmente soubéssemos, "o xadrez certamente perderia completamente o caráter de jogo". Uma sólida base para a teoria de jogos foi desenvolvida no original trabalho *Theory of Games and Economic Behavior* (von Neumann e Morgenstern, 1944), que incluía uma análise mostrando que alguns jogos *exigem* estratégias aleatórias (ou, pelo menos, imprevisíveis). Consulte o Capítulo 17 para obter mais informações.

John McCarthy concebeu a ideia de busca alfa-beta em 1956, embora não a tivesse publicado. O

programa de xadrez NSS (Newell *et al*., 1958) usava uma versão simplificada de alfa-beta; ele foi o primeiro programa de xadrez a usá-la. A poda alfa-beta foi descrita por Hart e Edwards (1961) e Hart *et al*. (1972). Alfa-beta também foi usada pelo programa de xadrez "Kotok-McCarthy", escrito por um aluno de John McCarthy (Kotok, 1962). Knuth e Moore (1975) provaram a exatidão de alfa-beta e analisaram sua complexidade de tempo. Pearl (1982b) demonstra que alfa-beta é assintoticamente ótima entre todos os algoritmos de busca de árvores de jogos de profundidade fixa.

Várias tentativas foram feitas para superar os problemas com a "abordagem-padrão" esboçados na Seção 5.8. O primeiro algoritmo de busca heurística não exaustiva com algum embasamento teórico provavelmente foi o B* (Berliner, 1979), que tentava manter limites intervalares sobre o valor possível de um nó na árvore de jogo, em vez de dar a ele uma única estimativa de valor pontual. Nós folhas são selecionados para expansão em uma tentativa de aprimorar os limites de nível superior até um único movimento se mostrar "claramente melhor". Palay (1985) estende a ideia de B*, utilizando distribuições de probabilidades sobre valores no lugar de intervalos. A busca de número de conspiração de David McAllester (1988) expande nós de folhas que, pela alteração de seus valores, poderiam fazer o programa preferir um novo movimento na raiz. O MGSS* (Russell e Wefald, 1989) usa as técnicas de teoria da decisão do Capítulo 16 para estimar o valor da expansão de cada folha em termos da melhoria esperada na qualidade da decisão na raiz. Ele superou um algoritmo alfa-beta em Othello, apesar de buscar um número de nós uma ordem de magnitude menor. Em princípio, a abordagem do MGSS* é aplicável ao controle de qualquer forma de deliberação.

Em muitos aspectos, a busca alfa-beta é a análoga para dois jogadores da busca em profundidade ramificada e limitada, dominada por A* no caso de um único agente. O algoritmo SSS* (Stockman, 1979) pode ser visualizado como um A* de dois jogadores e nunca expande mais nós que alfa-beta para chegar à mesma decisão. Os requisitos de memória e a sobrecarga computacional da fila tornam impraticável o SSS* em sua forma original, mas foi desenvolvida uma versão que necessita de espaço linear a partir do algoritmo RBFS (Korf e Chickering, 1996). Plaat *et al*. (1996) desenvolveram uma nova visão do SSS* como uma combinação de alfa-beta e tabelas de transposição, mostrando como superar as desvantagens do algoritmo original e desenvolvendo uma nova variante chamada MTD(*f*) que foi adotada por vários programas importantes.

D. F. Beal (1980) e Dana Nau (1980, 1983) estudaram as deficiências do minimax aplicado a avaliações aproximadas. Eles mostraram que, sob certas suposições de independência sobre a distribuição de valores de folhas na árvore, o uso do minimax pode gerar valores na raiz que na realidade são *menos* confiáveis que o uso direto da própria função de avaliação. O livro de Pearl, *Heuristics* (1984), explica parcialmente esse paradoxo aparente e analisa muitos algoritmos de jogos. Baum e Smith (1997) propõem um substituto para o minimax baseado em probabilidades, mostrando que ele resulta em escolhas melhores em certos jogos. O algoritmo expectiminimax foi proposto por Donald Michie (1966). Bruce Ballard (1983) estendeu a poda alfa-beta para cobrir árvores com nós de acaso e Hauk (2004) reexaminou esse trabalho e proporcionou resultados empíricos.

Koller e Pfeffer (1997) descreveram um sistema para resolver completamente jogos parcialmente observáveis. O sistema é bastante geral, manipulação de jogos cuja estratégia ótima exige movimentos aleatórios e jogos que são mais complexos do que aqueles manipulados por qualquer sistema anterior. Ainda assim, não pode manipular jogos tão complexos como pôquer, *bridge* e

Kriegspiel. Franco *et al.* (1998) descreveram diversas variantes da busca de Monte Carlo, incluindo uma em que o MIN tem informação completa, mas o MAX, não. Entre os jogos determinísticos, parcialmente observáveis, o Kriegspiel recebeu mais atenção. Ferguson demonstrou estratégias randomizadas deduzidas à mão (*hand-derived*) para ganhar do Kriegspiel com um bispo e o cavalo (1992) ou dois bispos (1995) contra um rei. Os primeiros programas Kriegspiel concentravam-se em encontrar xeque-mates de final de jogo e executavam a busca E-OU no espaço do estado de crença (Sakuta e Iida, 2002; Bolognesi e Ciancarini, 2003). Algoritmos de estado de crença incremental habilitaram xeque-mates de meio de jogo muito mais complexos de ser encontrados (Russell e Wolfe, 2005; Wolfe e Russell, 2007), mas a estimativa de estado eficiente continua a ser o principal obstáculo para jogo geral efetivo (Parker *et al.*, 2005).

O **xadrez** foi uma das primeiras tarefas realizadas em IA, com esforços iniciais por muitos dos pioneiros da computação, incluindo Konrad Zuse em 1945, Norbert Wiener em seu livro *Cybernetics*(1948) e Alan Turing em 1950 (veja Turing *et al.*, 1953). Mas foi o artigo de Claude Shannon, *Programing a Computer for Playing Chess* (1950), que teve o mais completo conjunto de ideias, descrevendo uma representação de posições do tabuleiro, uma função de avaliação, a busca de quiescência e algumas ideias de seletiva (não exaustiva) busca de jogo de árvore. Slater (1950) e os comentaristas de seu artigo também exploraram as possibilidades para o jogo de xadrez de computador.

D.G. Prinz (1952) concluiu um programa que resolvia problemas de final de jogo de xadrez, mas que não jogou um jogo completo. Stan Ulam e um grupo em Los Alamos National Lab produziram um programa que jogava xadrez em um tabuleiro de 6 × 6 sem bispos (Kister *et al.*, 1957). Podia buscar quatro jogadas profundas em cerca de 12 minutos. Alex Bernstein escreveu o primeiro programa documentado para jogar um jogo completo de xadrez-padrão (Bernstein e Roberts, 1958).[5]

A primeira partida de xadrez em computador apresentou o programa de Kotok-McCarthy do MIT (Kotok, 1962), e o programa ITEP escrito em meados dos anos 1960 no Institute of Theoretical and Experimental Physics em Moscou (Adelson-Velsky *et al.*, 1970). Essa partida intercontinental foi realizada por telégrafo. Terminou com uma vitória de 3×1 para o programa ITEP, em 1967. O primeiro programa de xadrez a competir com sucesso com os seres humanos foi o MacHack-6 do MIT (Greenblatt *et al.*, 1967). Sua taxa Elo de cerca de 1.400 foi bem acima do nível iniciante de 1.000.

O Prêmio Fredkin, criado em 1980, ofereceu prêmios por metas progressivas no jogo de xadrez. O Belle, que atingiu uma classificação de 2.250 (Condon e Thompson, 1982), ganhou um prêmio de US$5.000,00 pelo primeiro programa a conseguir uma classificação *master*. O prêmio de US$10.000 para o primeiro programa a conseguir uma classificação USCF (United States Chess Federation) de 2.500 (perto do nível *grandmaster*) foi atribuído ao DEEP THOUGHT (Hsu *et al.*, 1990), em 1989. O Deep Blue (Campbell *et al.*, 2002;. Hsu, 2004) recebeu o grande prêmio de US$100.000,00, por sua vitória referencial sobre o campeão mundial Garry Kasparov em um jogo de exibição de 1997. Kasparov escreveu:

> O jogo decisivo da partida foi o jogo 2, que deixou uma cicatriz em minha memória […] vimos algo que foi bem além de nossas expectativas mais otimistas do quanto um computador seria capaz de prever as consequências posicionais em longo prazo das suas decisões. A máquina recusou-se a

se mover para uma posição que tinha uma vantagem decisiva no curto prazo — mostrando um senso de perigo muito humano. (Kasparov, 1997)


(a)


(b)

Figura 5.15 Pioneiros no xadrez de computador: (a) Herbert Simon e Allen Newell, os desenvolvedores do programa NSS (1958); (b) John McCarthy e o programa Kotok-McCarthy em uma IBM 7090 (1967).

Provavelmente, a descrição mais completa de um programa de xadrez moderno foi fornecida por Ernst Heinz (2000), cujo programa DARKTHOUGHT foi o programa de PC não comercial mais bem classificado durante o campeonato mundial em 1999.

Nos últimos anos, os programas de xadrez estão se sobrepondo até mesmo aos melhores seres humanos do mundo. Entre 2004 e 2005, o HYDRA derrotou o grande mestre Eigen Vladimirov por 3,5×0,5, o campeão mundial Ruflam Ponomariov por 2×0, e o sétimo do *ranking* Michael Adams por 5,5×0,5. Em 2006, o DEEP FRITZ bateu o campeão mundial Vladimir Kramnik por 4×2 e, em 2007, o RYBKA venceu diversos grandes mestres em jogos em que ele deu chances (como um peão) para os jogadores humanos. A partir de 2009, a mais alta classificação Elo já registrada foi de 2.851 por Kasparov. O HYDRA (Donninger e Lorenz, 2004) foi classificado em algum lugar entre 2.850 e 3.000, principalmente com base em sua derrota por Michael Adams. O programa RYBKA foi classificado entre 2.900 e 3.100, mas isso foi baseado em um pequeno número de jogos e não é considerado confiável. Ross (2004) mostrou como os seres humanos aprenderam a explorar algumas das fraquezas dos programas de computador.

O **jogo de damas**, em vez do xadrez, foi o primeiro dos jogos clássicos disputado inteiramente por um computador. Christopher Strachey (1952) escreveu o primeiro programa funcional para jogo de damas. Com início em 1952, Arthur Samuel, da IBM, trabalhando em seu tempo livre, desenvolveu um programa de damas que aprendeu sua própria função de avaliação, jogando consigo mesmo milhares de vezes (Samuel, 1959, 1967). Descreveremos essa ideia com mais detalhes no Capítulo 21. O programa de Samuel começou como um novato, mas depois de apenas alguns dias o autojogo tinha melhorado além do próprio nível de Samuel. Em 1962, derrotou Robert Nealy, campeão em “damas cegas”, através de um erro de sua parte. Quando se considera que os equipamentos de computação de Samuel (um IBM 704) tinham 10.000 palavras na memória principal, uma fita magnética para armazenamento de longo prazo e um processador de 0,000001 GHz, a vitória é

considerada uma grande realização.

O desafio iniciado por Samuel foi retomado por Jonathan Schaeffer, da Universidade de Alberta. Seu programa CHINOOK ficou em segundo lugar no Aberto dos Estados Unidos de 1990 e recebeu o direito de desafio para o campeonato mundial. Em seguida, enfrentou um problema chamado Marion Tinsley. O Dr. Tinsley tinha sido campeão do mundo há mais de 40 anos, perdendo apenas três jogos em todo esse tempo. Na primeira partida contra o Chinook, Tinsley sofreu suas quarta e quinta derrotas, mas venceu a partida por 20,5×18,5. Uma revanche no campeonato mundial de 1994 terminou prematuramente quando Tinsley teve que se retirar por motivos de saúde. O CHINOOK tornou-se oficialmente o campeão do mundo. Schaeffer continuou construindo seu banco de dados de finais de jogos e, em 200, "resolveu" o jogo de damas (Schaeffer *et al*., 2007; Schaeffer, 2008). Isso tinha sido previsto por Richard Bellman (1965). No documento que introduziu a abordagem de programação dinâmica para análise retrógrada, ele escreveu: "No jogo de damas, o número de movimentos possíveis em qualquer situação dada é tão pequeno que podemos esperar com confiança uma solução digital computacional completa para o problema de jogada ótima." No entanto, Bellman não estimou plenamente o tamanho da árvore do jogo de damas. Existem cerca de 500 quatrilhões de posições. Depois de 18 anos de computação em um *cluster* de 50 ou mais máquinas, a equipe de Jonathan Schaeffer completou uma tabela de final de jogo para todas as posições do jogo de damas com 10 ou menos peças: mais de 39 trilhões de entradas. A partir daí, foram capazes de fazer busca alfa-beta adiantada para derivar uma política que prova que o jogo de damas é de fato um empate com o melhor jogo de ambos os lados. Observe que esta é uma aplicação de busca bidirecional (Seção 3.4.6). A construção de uma tabela de final de jogo para todos os jogos de damas seria impraticável: seria necessário um bilhão de gigabytes de armazenamento. A busca sem qualquer tabela também seria impraticável: a árvore de busca tem cerca de 8^{47} posições, e levaria milhares de anos de pesquisa com a tecnologia atual. Só uma combinação de busca inteligente, dados de final de jogo e uma queda no preço dos processadores e da memória poderia resolver o jogo de damas. Assim, o jogo de damas juntou-se com o Qubic (Patashnik, 1980), o Connect Four (Allis, 1988) e o Nine-Men's Morris (Gasser, 1998) como jogos que foram resolvidos por análise de computador.

O **gamão**, um jogo de azar, foi analisado matematicamente por Gerolamo Cardano (1663), mas foi tido como jogo de computador apenas no final de 1970, primeiro com o programa BKG (Berliner, 1980b); ele usou uma função de avaliação complexa, construída manualmente e pesquisada apenas à profundidade 1. Foi o primeiro programa a derrotar um campeão mundial humano em um jogo clássico maior (Berliner, 1980a). Berliner reconheceu prontamente que o BKG tinha muita sorte com os dados. O TD-Gammon de Gerry Tesauro (1995) jogou consistentemente em campeonato em nível mundial. O programa BGBLITZ foi o vencedor da Computer Olympiad de 2008.

O **Go** é um jogo determinístico, mas o grande fator de ramificação o torna um desafio. As questões-chave e a literatura prévia em computação com o Go foram resumidas por Bouzy e Cazenave (2001) e Muller (2002). Até 1997 não existiam programas de Go competentes. Hoje, os melhores programas jogam a *maioria* dos seus lances em nível de mestre; o único problema é que no decorrer de um jogo eles costumam fazer pelo menos um erro grave que permite que um oponente forte vença. Considerando que a busca alfa-beta reina na maioria dos jogos, muitos programas Go recentes têm adotado os métodos de Monte Carlo com base em esquema UCT (limite de confiança superior em árvores) (Kocsis e Szepesvari, 2006). O programa Go mais potente como o de 2009 é o

MOGO de Gelly e Silver (Wang e Gelly, 2007; Gelly e Silver, 2008). Em agosto de 2008, o MOGO marcou uma vitória surpreendente contra o profissional de alto nível Myungwan Kim, embora com o MoGo recebendo uma vantagem de nove pedras (o equivalente a uma vantagem da rainha no xadrez). Kim estimou a resistência de MoGo em 2-3 dan, o nível baixo de amador avançado. Para essa partida, o Mogo foi executado em um supercomputador com processador 800 de 15 teraflop (1.000 vezes o Deep Blue). Algumas semanas mais tarde, o Mogo, com apenas uma desvantagem de cinco pedras, ganhou contra um profissional por 6 dan. Na forma 9 × 9 do Go, o MoGo está aproximadamente no nível profissional 1 dan. Os avanços rápidos, provavelmente como experimentação, continuam com novas formas da busca de Monte Carlo. O *Boletim Go Computer*, publicado pela Computer Go Association, descreve a evolução atual.

Bridge: Smith *et al*. (1998) relataram sobre como o seu programa baseado em planejamento ganhou o campeonato de *bridge* por computador de 1998 e descreveram (Ginsberg, 2001) como o seu programa GIB, com base na simulação de Monte Carlo, venceu o campeonato seguinte de computador, e surpreendentemente bem, contra jogadores humanos e conjuntos de problemas-padrão de livro. Em 2001-2007, o campeonato de *bridge* por computador foi vencido cinco vezes por JACK e duas vezes por WBRIDGE. Não existem artigos acadêmicos de nenhum dos dois explicando sua estrutura, mas os rumores é que ambos usaram a técnica de Monte Carlo, que foi proposta pela primeira vez para o jogo de *bridge* por Levy (1989).

Palavras cruzadas: Brian Sheppard (2002), seu criador, forneceu uma boa descrição de um programa de topo, o Maven. A geração de movimentos de pontuação mais alta é descrita por Gordon (1994), e a modelagem dos adversários foi coberta por Richards e Amir (2007).

Futebol (Kitano *et al*., 1997b; Visser *et al*., 2008) e **bilhar** (Lam e Greenspan, 2008; Archibald *et al*., 2009) e outros jogos estocásticos com espaço contínuo de ações estão começando a atrair a atenção em IA, tanto na simulação como com jogadores robôs físicos.

Competições de jogos de computador ocorrem anualmente, e os artigos aparecem em uma variedade de locais. Os anais da conferência de nome bastante ilusório, *Heuristic Programming in Artificial Intelligence*, relatam as Computer Olympiads, que incluem grande variedade de jogos. O General Game Competition (Love *et al*., 2006) testa programas que devem aprender a jogar determinado jogo desconhecido apenas com uma descrição lógica das suas regras. Também existem diversas coleções editadas de artigos importantes sobre pesquisa na área de jogos (Levy, 1988a, 1988b; Marsland e Schaeffer, 1990). A International Computer Chess Association (ICCA), fundada em 1977, publica o periódico trimestral *ICGA Journal* (antigamente denominado *ICCA Journal*). Foram publicados artigos importantes na antologia em série *Advances in Computer Chess*, começando com o artigo de Clarke (1977). O volume 134 do periódico *Artificial Intelligence* (2002) contém descrições de programas de última geração para xadrez, Othello, Hex, *shogi*, Go, gamão, pôquer, Scrabble™ e outros jogos. Desde 1998 tem sido realizada uma conferência bienal, *Computers and Games*.

EXERCÍCIOS

5.1 Suponha que você tenha um oráculo, *OM*(*s*), que prevê corretamente o lance do oponente em

qualquer estado. Utilizando isso, formule a definição de um jogo como um problema de busca (agente único). Descreva um algoritmo para encontrar o lance ótimo.

5.2 Considere o problema de resolver o quebra-cabeça de oito peças.

a. Dê uma formulação completa do problema no estilo do Capítulo 3.

b. Qual o tamanho do espaço de estados alcançável? Forneça uma expressão numérica exata.

c. Suponha que desenvolvamos um problema adversário da seguinte forma: os dois jogadores se revezam em lance; uma moeda é arremessada para determinar o quebra-cabeça no qual o movimento deve ser feito nessa vez; e o vencedor será o primeiro a resolver um quebra-cabeça. Qual algoritmo pode ser usado para a escolha de um lance nesse cenário?

d. Dê uma prova informal de que eventualmente alguém vai vencer se ambos jogarem perfeitamente.

5.3 Imagine que, no Exercício 3.3, um dos amigos queira evitar o outro. O problema então se torna um jogo de dois jogadores de **perseguição e evasão**. Assumamos agora que os jogadores se movem por vez. O jogo só termina quando os jogadores estão no mesmo nó; o resultado final para o perseguidor é menos o tempo total necessário (o evasor "ganha" por nunca perder). A Figura 5.16 mostra um exemplo.

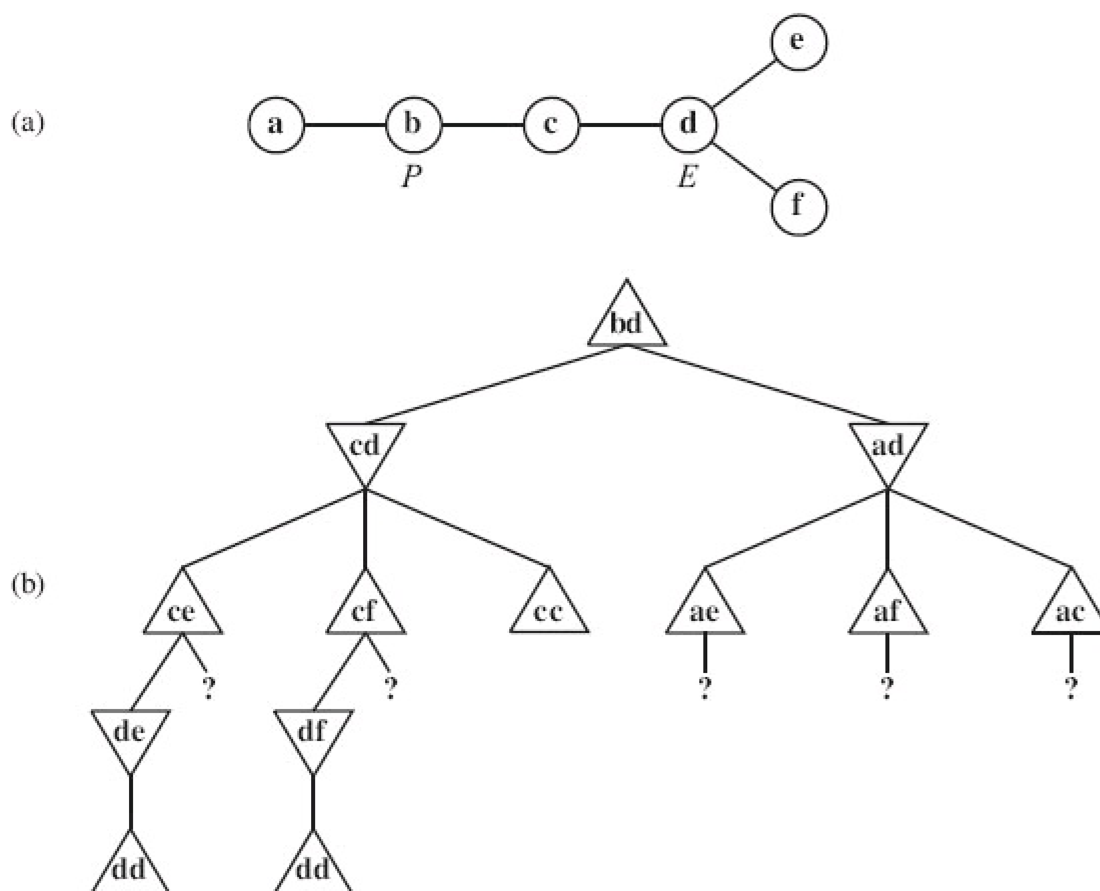


Figura 5.16 (a) Um mapa onde o custo de cada aresta é 1. Inicialmente, o perseguidor *P* está no nó **b** e o evasor *E* está no nó **d**. (b) Uma árvore de jogo parcial para esse mapa. Cada nó é rotulado com as posições *P* e *E*. *P* move-se em primeiro lugar. Ainda não foram explorados os ramos marcados com "?".

a. Copie a árvore de jogo e marque os valores dos nós terminais.

b. Ao lado de cada nó interno, escreva o fato mais forte que você pode inferir sobre o seu valor (um número, uma ou mais desigualdades, tais como “≥14” ou um “?”).

c. Abaixo de cada ponto de interrogação, escreva o nome do nó atingido por esse ramo.

d. Explique como um limite sobre o valor dos nós em (c) pode ser derivado da consideração do comprimento do caminho mais curto no mapa e deduza tais limites para esses nós. Lembre-se do custo para chegar a cada folha, bem como do custo para resolvê-la.

e. Suponha agora que a árvore como dada, com os limites de folha de (d), seja avaliada da esquerda para a direita. Circule os nós “?” que *não* precisam mais ser expandidos, dados os limites da parte (d), e risque aqueles que não precisam ser considerados.

f. Você pode provar algo genérico sobre quem vai ganhar o jogo em um mapa que é uma árvore?

5.4 Descreva ou implemente descrições de estados, geradores de movimentos, testes de término, funções utilidade e funções de avaliação para um ou mais dos seguintes jogos estocásticos: Monopoly, Scrabble, *bridge* com determinado acordo ou *Texas hold’em poker* (variante do pôquer).

5.5 Descreva e implemente um ambiente de jogos de *tempo real*, de *vários participantes*, em que o tempo faça parte do estado do ambiente e no qual os jogadores recebam alocações de tempo fixas.

5.6 Descreva o quanto a abordagem-padrão para jogos se aplicaria bem a jogos como tênis, bilhar e *croquet*, que ocorrem em um espaço de estados físicos contínuo.

5.7 Prove a seguinte afirmativa: para toda árvore de jogo, a utilidade obtida por MAX usando decisões de minimax contra um MIN não ótimo nunca será mais baixa que a utilidade obtida no jogo contra um MIN ótimo. Você poderia apresentar uma árvore de jogo em que MAX pudesse atuar ainda melhor usando uma estratégia *ótima* contra um MIN não ótimo?

5.8 Considere o jogo de dois jogadores descrito na Figura 5.17.

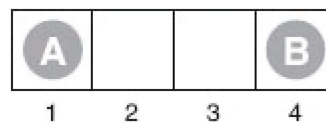


Figura 5.17 Posição inicial de um jogo simples. O jogador *A* joga primeiro. Os dois jogadores se revezam na movimentação e cada jogador deve mover sua ficha para um espaço adjacente aberto em qualquer sentido. Se o oponente ocupar um espaço adjacente, um jogador pode saltar sobre um oponente até o próximo espaço aberto, se houver (por exemplo, se *A* estiver em 3 e *B* estiver em 2, A poderá voltar a 1). O jogo termina quando um jogador chegar à extremidade oposta do tabuleiro. Se o jogador *A* alcançar o espaço 4 primeiro, o valor do jogo para *A* será +1; se o jogador *B* alcançar o espaço 1 primeiro, o valor do jogo para *A* será −1.

a. Desenhe a árvore de jogo completa, usando as convenções a seguir:

– Escreva cada estado como (s_A, s_B), onde s_A e s_B denotam as posições das fichas.
– Coloque cada estado terminal em um quadrado e escreva o valor de seu jogo em um círculo.
– Insira os *estados de ciclo* (estados que já aparecem no caminho para a raiz) em quadrados duplos. Tendo em vista que o valor não está claro, identifique cada um com um símbolo “?”

dentro de um círculo.

b. Agora marque cada nó com seu valor minimax propagado de volta (também em um círculo). Explique como você tratou os valores “?” e por quê.

c. Explique por que o algoritmo minimax-padrão falharia nessa árvore de jogo e faça um resumo de como você poderia corrigi-lo, baseando-se em sua resposta ao item (b). Seu algoritmo modificado oferece decisões ótimas para todos os jogos com ciclos?

d. Esse jogo de quatro quadrados pode ser generalizado para n quadrados, para qualquer $n > 2$. Prove que A vence se n é par e perde se n é ímpar.

5.9 Este problema exercita os conceitos básicos do jogo usando o jogo da velha (zeros e cruzes) como exemplo. Nós definimos X_n como o número de linhas, colunas ou diagonais com exatamente n X e não O. Da mesma forma, On é o número de linhas, colunas ou diagonais com apenas n O. A função utilidade atribui +1 a qualquer posição com $X_3 = 1$ e −1 em qualquer posição com $O_3 = 1$. Todas as outras posições terminais têm utilidade 0. Para posições não terminais, utilizamos uma função linear de avaliação definida como $Eval(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$.

a. Existem cerca de quantos jogos da velha possíveis?

b. Mostre toda a árvore de jogo começando de um tabuleiro vazio até a profundidade 2 (isto é, um X e um O na tabuleiro), levando a simetria em conta.

c. Marque em sua árvore as avaliações de todas as posições com profundidade 2.

d. Utilizando o algoritmo minimax, marque em sua árvore os valores propagados para as posições à profundidade 1 e 0, e utilize esses valores para escolher a melhor jogada da partida.

e. Circule os nós à profundidade 2 que *não* seriam avaliados se a poda alfa-beta fosse aplicada, assumindo que os nós são gerados na ordem ótima para a poda alfa-beta.

5.10 Considere a família de jogos da velha generalizada, definida da seguinte forma. Cada jogo em particular é especificado por um conjunto S de *quadrados* e uma coleção W de *posições vencedoras*. Cada posição vencedora é um subconjunto de S. Por exemplo, no jogo da velha padrão, S é um conjunto de nove quadrados e W é uma coleção de oito subconjuntos de W: as três linhas, as três colunas e as duas diagonais. Em outros aspectos, o jogo é idêntico ao jogo da velha-padrão. A partir de um tabuleiro vazio, os jogadores alternam-se colocando suas marcas em um quadrado vazio. Um jogador que marcar cada quadrado em uma posição vencedora ganhará o jogo. Será empate se todos os quadrados forem marcados e ninguém ganhar o jogo.

a. Seja $N = |S|$ o número de quadrados. Dê um limite superior do número de nós em uma árvore de jogo completa para o jogo da velha comum em função de N.

b. Dê um limite inferior para o tamanho da árvore de jogo para o pior caso, onde $W = \{\ \}$.

c. Proponha uma função de avaliação plausível que possa ser utilizada para qualquer exemplo de jogo da velha comum. A função pode depender de S e W.

d. Assuma que seja possível gerar um novo tabuleiro e verifique se a instrução de máquina $100N$ é uma posição vencedora, assumindo um processador de 2 gigahertz. Ignore as limitações de memória. Utilizando a sua estimativa em (a), aproximadamente com que tamanho uma árvore de jogo pode ser completamente resolvida por alfa-beta em um segundo de tempo de CPU, em um

minuto, em uma hora?

5.11 Desenvolva um programa de jogo genérico capaz de jogar uma variedade de jogos.

a. Implemente geradores de movimentação e funções de avaliação para um ou mais dos seguintes jogos: Kalah, Otelo, damas e xadrez.

b. Construa um agente de jogo genérico alfa-beta.

c. Compare o efeito de aumentar a profundidade da busca melhorando a ordem dos lances e melhorando a função de avaliação. O quão próximo chegará o seu fator efetivo de ramificação para o caso ideal de ordenação de movimentação perfeita?

d. Implemente um algoritmo de busca seletiva, tal como B* (Berliner, 1979), busca de número de conspiração (McAllester, 1988) ou MGSS * (Russell e Wefald, 1989) e compare seu desempenho para A*.

5.12 Descreva como os algoritmos minimax e alfa-beta se alteram para **jogos de soma diferente de zero** com dois jogadores, em que cada jogador tem sua própria função utilidade e as funções utilidades são conhecidas pelos dois jogadores. Suponha que cada jogador conheça a função utilidade do outro. Se não houver restrições sobre as duas utilidades terminais, é possível qualquer nó ser podado por alfa-beta? O que acontecerá se as funções utilidade do jogador em qualquer estado for a soma de um número entre as constantes −k e k, tornando o jogo quase de soma zero?

5.13 Desenvolva uma prova formal da correção da poda alfa-beta. Para isso, considere a situação mostrada na Figura 5.18. A questão é se devemos podar ou não o nó n_j, um nó de máximo descendente do nó n_1. A ideia básica é podá-lo se e somente se for possível mostrar que o valor minimax de n_1 é independente do valor de n_j.

Figura 5.18 Situação quando se considera se o nó n_j deve ser podado.

a. A Modalidade n_1 assume o valor mínimo entre seus filhos: $n_1 = \min(n_2, n_{21},\ldots,n_{2b2})$. Encontre uma expressão semelhante para n_2 e, consequentemente, uma expressão para n_1 em termos de nj.

b. Seja l_i o valor mínimo (ou máximo) dos nós à *esquerda* do nó n_i na profundidade i, cujo valor

minimax já é conhecido. De modo semelhante, seja r_i o valor mínimo (ou máximo) dos nós não explorados à direita de n_i na profundidade i. Reescreva sua expressão para n_1 em termos dos valores l_i e r_i.

c. Agora, reformule a expressão com a finalidade de mostrar que, para afetar n_1, o valor de n_j não deve exceder certo limite derivado dos valores de l_i.

d. Repita o processo para o caso em que n_j é um nó de MIN.

5.14 Prove que a poda alfa-beta leva o tempo de $O(2^{m/2})$ com ordenação de lance ótimo, onde m é a profundidade máxima da árvore de jogo.

5.15 Vamos supor que você tenha um programa de xadrez capaz de avaliar 10 milhões de nós por segundo. Decida-se por uma representação compacta de um estado de jogo que será armazenada em uma tabela de transposição. Quantas entradas aproximadamente você poderá colocar em uma tabela de 2 gigabytes na memória? Isso será suficiente para os três minutos de busca alocados a um único movimento? Quantos acessos à tabela você poderá efetuar no tempo que levaria para realizar uma única avaliação? Agora, suponha que a tabela de transposição seja maior do que é possível caber na memória. Aproximadamente quantas avaliações você poderia efetuar no tempo necessário para realizar um único acesso a disco com hardware de disco-padrão?

5.16 Esta questão considera a poda em jogos com nós de acaso. A Figura 5.19 mostra a árvore de jogo completa para um jogo trivial. Suponha que os nós folha estejam para ser avaliados na ordem da esquerda para a direita e que, antes de um nó folha ser avaliado, não sabemos nada sobre o seu valor — a faixa de valores possíveis é $-\infty$ até ∞.

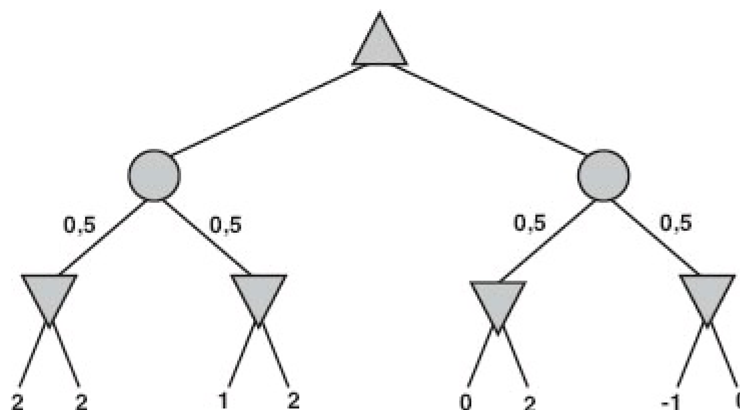


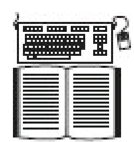
Figura 5.19 A árvore de jogo completa para um jogo trivial com nós de acaso.

a. Copie a figura, marque o valor de todos os nós internos e indique a melhor jogada na raiz com uma seta.

b. Dados os valores das primeiras seis folhas, precisamos avaliar a sétima e oitava folhas? Dados os valores das sete primeiras folhas, precisamos avaliar a oitava folha? Explique suas respostas.

c. Suponha que os valores do nó folha estejam entre −2 e 2, inclusive. Após as duas primeiras folhas serem avaliadas, qual é o intervalo de valor para o nó de acaso da esquerda?

d. Circule todas as folhas que não precisam ser avaliadas sob o pressuposto em (c).



5.17 Implemente o algoritmo expectiminimax e o algoritmo *-alfa-beta, descrito por Ballard (1983), para podar árvores de jogo com nós de acaso. Experimente-os em um jogo como o gamão e meça a eficiência de poda do algoritmo *-alfa-beta.

5.18 Prove que, com uma transformação linear positiva de valores de folha (isto é, a transformação de um valor x em $ax + b$ onde $a > 0$), a escolha do movimento permanece inalterada em uma árvore de jogo, mesmo quando existem nós de acaso.

5.19 Considere o procedimento a seguir para escolher movimentos em jogos com nós de acaso:

- Gere algumas sequências de lançamentos de dados (digamos, 50) descendo até uma profundidade apropriada (digamos, 8).
- Com lançamentos de dados conhecidos, a árvore de jogo se torna determinística. Para cada sequência de lançamentos de dados, resolva a árvore de jogo determinística resultante usando alfa-beta.
- Utilize os resultados para estimar o valor de cada movimento e escolher o melhor.
- Esse procedimento funcionará bem? Por quê (ou por que não)?

5.20 A seguir, uma árvore "max" consiste apenas em nós max, enquanto uma árvore "expectimax" consiste em um nó max na raiz com camadas alternadas ao acaso e nós max. Nos nós de acaso, todas as probabilidades resultantes são diferentes de zero. O objetivo é *encontrar o valor da raiz* com uma busca de profundidade limitada.

a. Assumindo que os valores da folha são finitos mas ilimitados, será possível a poda (como em alfa-beta) em uma árvore max? Dê um exemplo ou explique por que não.

b. A poda é sempre possível em uma árvore expectimax nas mesmas condições? Dê um exemplo ou explique por que não.

c. Se os valores da folha são não negaivos, a poda é sempre possível em uma árvore max? Dê um exemplo ou explique por que não.

d. Se os valores da folha são não negativos, a poda é sempre possível em uma árvore expectimax? Dê um exemplo ou explique por que não.

e. Se os valores da folha são restritos na faixa [0, 1], a poda é sempre possível em uma árvore max? Dê um exemplo ou explique por que não.

f. Se os valores da folha são restritos na faixa [0, 1], a poda é sempre possível em uma árvore expectimax?

g. Considere os resultados de um nó de acaso em uma árvore expectimax. Qual das seguintes ordens de avaliação é mais provável de produzir oportunidades de poda?

(i) Menor probabilidade em primeiro lugar

(ii) Maior probabilidade em primeiro lugar

(iii) Não faz qualquer diferença

5.21 Quais das seguintes alternativas são verdadeiras e quais são falsas? Dê breves explicações.

a. Em um jogo totalmente observável, de revezamento, de soma zero, entre dois jogadores perfeitamente racionais, não ajuda o primeiro jogador saber que estratégia o segundo jogador está usando, isto é, o lance do segundo jogador é baseado no lance do primeiro jogador.

b. Em um jogo totalmente observável, de revezamento, de soma zero, entre dois jogadores perfeitamente racionais, não ajuda o primeiro jogador saber que lance o segundo jogador fará, dado o lance do primeiro jogador.

c. Um agente de gamão perfeitamente racional nunca perde.

5.22 Considere cuidadosamente a interação de eventos de acaso e a informação parcial em cada um dos jogos do Exercício 5.4.

a. Para qual deles o modelo expectiminimax padrão é apropriado? Implemente o algoritmo e execute-o em seu agente de jogos, com modificações apropriadas para o ambiente de jogos.

b. Para quais deles o esquema descrito no Exercício 5.19 seria apropriado?

c. Descreva como você poderia lidar com o fato de que, em alguns jogos, os jogadores não têm o mesmo conhecimento do estado corrente.

---

[1] Ambientes com muitos agentes são mais bem visualizados como **economias**, em vez de jogos.

[2] É óbvio que isso não pode ser realizado com perfeição; caso contrário, a função de ordenação poderia ser utilizada para se jogar um jogo perfeito!

[3] Às vezes, o estado de crença vai se tornar muito grande para representar apenas uma lista de estados de tabuleiro, mas vamos ignorar essa questão por ora; os Capítulos 7 e 8 sugerem um método para representar compactamente os estados de crença muito grandes.

[4] Blefar — apostar que uma mão é boa, mesmo quando não é — é uma parte essencial da estratégia do pôquer.

[5] Um programa russo, o BESM, pode ter antecipado o programa Bernstein.

CAPÍTULO 6

Problemas de satisfação de restrições

Em que vemos como o tratamento de estados como algo mais que apenas pequenas caixas-pretas nos leva à criação de ampla variedade de novos e poderosos métodos de busca e a uma compreensão mais profunda da estrutura e da complexidade dos problemas.

Os Capítulos 3 e 4 exploraram a ideia de que os problemas podem ser resolvidos buscando-se em um espaço de **estados**. Esses estados podem ser avaliados por heurísticas específicas de domínios e testados para se verificar se eles são estados objetivo. Porém, do ponto de vista do algoritmo de busca, cada estado é **atômico, ou indivisível** — uma caixa-preta, sem estrutura interna.

Este capítulo descreve uma maneira de resolver uma grande variedade de problemas de forma mais eficiente. Utilizaremos uma **representação fatorada** para cada estado: um conjunto de variáveis, cada qual com um valor. O problema será resolvido quando cada variável tiver um valor que satisfaça todas as restrições sobre a variável. Um problema assim descrito é chamado de **problema de satisfação de restrição** ou PSR.

Os algoritmos de busca PSR aproveitam a estrutura de estados e utilizam heurísticas de *propósito geral* em vez de heurísticas *específicas de problemas* para permitir a solução de problemas complexos. A ideia principal é eliminar grande parte do espaço de busca de uma só vez através da identificação de combinações de variável/valor que violam as restrições.

6.1 DEFINIÇÃO DE PROBLEMAS DE SATISFAÇÃO DE RESTRIÇÕES

Um problema de satisfação de restrição consiste em três componentes, X, D e C:

X é um conjunto de variáveis, $\{X_1, \ldots, X_n\}$.
D é um conjunto de domínios, $\{D_1, \ldots, D_n\}$, um para cada variável.
C é um conjunto de restrições que especificam combinações de valores possíveis.

Cada domínio D_i consiste em um conjunto de valores possíveis, $\{v_1, \ldots, v_k\}$ para a variável X_i. Cada restrição C_i consiste em um par $\langle escopo, rel \rangle$, onde *escopo* é uma tupla de variáveis que participam da restrição e *rel* é uma relação que define os valores que essas variáveis podem assumir.

Uma relação pode ser representada como uma lista explícita de todas as tuplas de valores que satisfazem a restrição ou como uma relação abstrata que sustenta duas operações: testar se uma tupla é um membro da relação e enumerar os membros da relação. Por exemplo, se X_1 e X_2 têm o domínio {A, B}, então a restrição que exprime as duas variáveis deve ter valores diferentes que podem ser escritos como $\langle(X_1, X_2), [(A, B), (B, A)]\rangle$ ou como $\langle(X_1, X_2), X_1 \neq X_2\rangle$.

Para resolver um PSR, precisamos definir um espaço de estados e a noção de solução. Cada estado em um PSR é definido por uma **atribuição** de valores a algumas ou todas as variáveis, $\{X_i = v_i, X_j = v_j \ldots\}$. Uma atribuição que não viola quaisquer restrições é chamada de atribuição **consistente** ou legal. Uma **atribuição completa** é aquela em que cada variável é atribuída, e uma **solução** para um PSR é uma atribuição consistente e completa. A **atribuição parcial** é aquela que atribui valores para apenas algumas das variáveis.

6.1.1 Exemplo de problema: coloração de mapa

Vamos supor que, cansados da Romênia, estamos observando um mapa da Austrália, que mostra cada um de seus estados e territórios, como o da Figura 6.1(a). Recebemos a tarefa de colorir cada região de vermelho, verde ou azul, de tal modo que nenhuma região vizinha tenha a mesma cor. Para formular esse problema como um PSR, definimos as variáveis para representar as regiões

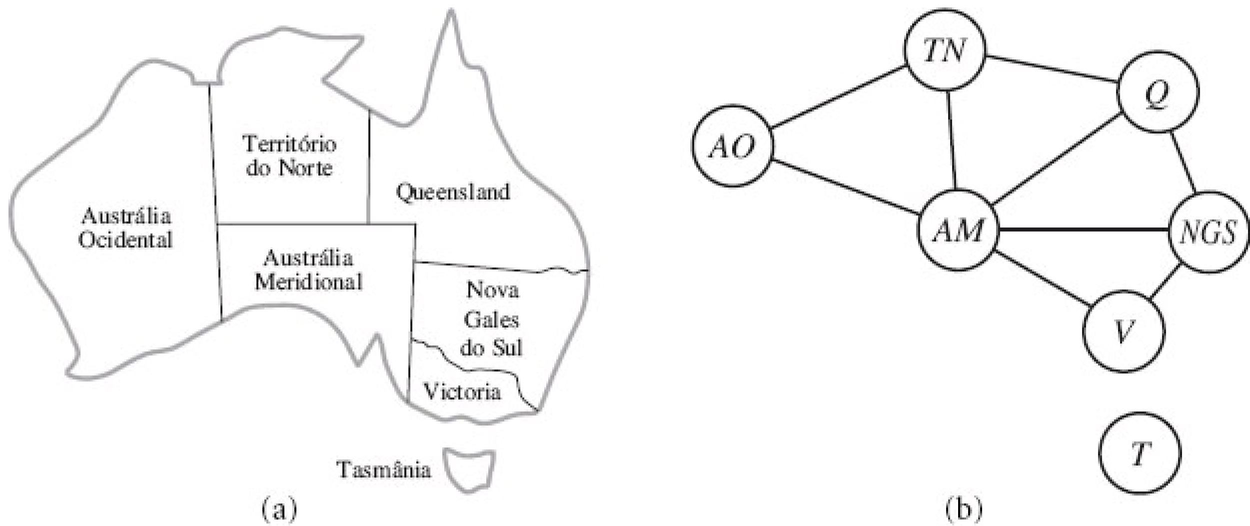


Figura 6.1 (a) Os principais estados e territórios da Austrália. A coloração desse mapa pode ser vista como um problema de satisfação de restrições (PSR). O objetivo é atribuir cores a cada região de modo que não haja regiões vizinhas com a mesma cor. (b) O problema de coloração de mapa representado como um grafo de restrições.

$$X = \{AO, TN, Q, NGS, V, AM, T\}.$$

O domínio de cada variável é o conjunto D_i = {*vermelho, verde, azul*}. As restrições exigem que regiões vizinhas tenham cores distintas. Como há nove lugares onde as regiões são fronteiriças, há nove restrições:

$$C = \{AM \neq AO, AM \neq TN, AM \neq Q, AM \neq NGS, AM \neq V,$$
$$AO \neq TN, TN \neq Q, Q \neq NGS, NGS \neq V\}.$$

Aqui estamos utilizando abreviaturas; $AM \neq AO$ é um atalho para $\langle (AM, AO), AM \neq AO \rangle$, onde $AM \neq AO$ por sua vez pode ser totalmente enumerado como

$$\{(vermelho, verde), (vermelho, azul), (verde, vermelho), (verde, azul), (azul, vermelho), (azul, verde)\}.$$

Existem muitas soluções possíveis para este problema, tais como:

$$\{AO = vermelho, TN = verde, Q = vermelho, NGS = verde, V = vermelho,$$
$$AM = azul, T = vermelho\}.$$

É útil visualizar um PSR como um **grafo de restrições**, como mostra a Figura 6.1(b). Os nós do grafo correspondem às variáveis do problema, e uma ligação conecta quaisquer duas variáveis que participem de uma restrição.

Por que formular um problema como um PSR? Uma razão é que os PSRs produzem uma representação natural para uma ampla variedade de problemas; se você já tem um sistema de resolução de PSR, é muitas vezes mais fácil utilizá-lo para resolver um problema do que projetar uma solução personalizada utilizando outra técnica de busca. Além disso, solucionadores de PSR podem ser mais rápidos do que buscadores de espaço de estados porque os solucionadores de PSR podem eliminar rapidamente grandes amostras do espaço de busca. Por exemplo, uma vez escolhido {AM = *azul*} no problema da Austrália, podemos concluir que nenhuma das cinco variáveis vizinhas pode assumir o valor *azul*. Sem tomar partido da propagação de restrições, um procedimento de busca teria que considerar $3^5 = 243$ atribuições para as cinco variáveis vizinhas; com propagação de restrições nunca teremos que considerar o *azul* como um valor, por isso temos apenas $2^5 = 32$ atribuições para verificar, uma redução de 87%.

Em busca de espaço de estados regulares podemos apenas perguntar: esse estado específico é um objetivo? Não? E esse aqui? Com PSRs, uma vez que descobrimos que uma atribuição parcial não é uma solução, podemos descartar imediatamente novos refinamentos de atribuição parcial. Além disso, podemos ver *por que* a atribuição não é uma solução — verificamos quais variáveis violam uma restrição — para que possamos centrar a atenção sobre as variáveis que importam. Como resultado, muitos problemas que são intratáveis para busca de espaço de estados regular podem ser rapidamente resolvidos quando formulados como um PSR.

6.1.2 Problema-exemplo: agendamento de tarefas

As fábricas têm o problema de fixar o valor de um dia de trabalho, sujeito a várias restrições. Na prática, muitos desses problemas são resolvidos com técnicas de PSR. Considere o problema de planejar a montagem de um carro. Todo o trabalho é composto de tarefas, e podemos modelar cada tarefa como uma variável, em que o valor de cada variável é o tempo em que começa a tarefa, expresso como um número inteiro de minutos. As restrições podem afirmar que uma tarefa deve ocorrer antes da outra, por exemplo, uma roda deve ser instalada antes que a calota seja colocada, e que apenas tais tarefas podem ser simultâneas. As restrições também podem especificar que uma

tarefa leva certo tempo para ser concluída.

Consideraremos uma pequena parte da montagem de automóveis, composta de 15 tarefas: instalar eixos (frente e trás), afixar quatro rodas (direita e esquerda, frente e trás), apertar as porcas em cada roda, afixar as calotas e inspecionar a montagem final. Podemos representar as tarefas com 15 variáveis:

$$X = \{Eixo_F, Eixo_T, Roda_{DF}, Roda_{EF}, Roda_{DT}, Roda_{ET}, Porcas_{DF}, Porcas_{EF}, Porcas_{DT}, Porcas_{ET}, Calota_{DF}, Calota_{EF}, Calota_{DT}, Calota_{ET}, inspecionar\}.$$

O valor de cada variável é a hora que a tarefa começa. Em seguida, representaremos as **restrições de precedência** entre tarefas individuais. Sempre que uma tarefa T_1 deve ocorrer antes da tarefa T_2 e a tarefa T_1 tem duração d_1 para acabar, adiciona-se uma restrição aritmética da forma

$$T_1 + d_1 \leq T_2.$$

No nosso exemplo, os eixos têm que estar no lugar antes de as rodas serem colocadas, e leva 10 minutos para instalar um eixo, de modo que escrevemos

$$Eixo_F + 10 \leq Roda_{DF};\ Eixo_F + 10 \leq Roda_{EF};$$
$$Eixo_T + 10 \leq Roda_{DT};\ Eixo_T + 10 \leq 10\ Roda_{LT}.$$

Em seguida, dizemos que, para cada roda, devemos afixar a roda (o que leva um minuto), em seguida, apertar as porcas (dois minutos) e, finalmente, colocar a calota (um minuto, mas não está ainda representado):

$$Roda_{DF} + 1 \leq Porcas_{DF};\ Porcas_{DF} + 2 \leq Calota_{DF};$$
$$Roda_{EF} + 1 \leq Porcas_{EF};\ Porcas_{EF} + 2 \leq Calota_{EF};$$
$$\mathrm{Roda}_{DT} + 1 \leq Porcas_{DT};\ Porcas_{DT} + 2 \leq Calota_{DT};$$
$$Roda_{LT} + 1 \leq Porcas_{LT};\ Porcas_{LT} + 2 \leq Calota_{LT}.$$

Suponha que tenhamos quatro trabalhadores para instalar as rodas, mas eles têm que compartilhar uma ferramenta que ajuda a colocar o eixo no lugar. Precisamos de uma **restrição disjuntiva** para dizer que o *EixoF* e o *EixoT* não devem ser sobrepostos no tempo: ou um ou outro deve vir em primeiro lugar:

$$(Eixo_F + 10 \leq Eixo_T)\ \mathbf{ou}\ (Eixo_T + 10 \leq Eixo_F).$$

Parece ser uma restrição mais complicada, combinando aritmética e lógica, mas ainda se reduz a um conjunto de pares de valores que o $Eixo_F$ e o $Eixo_F$ podem assumir.

Precisamos ainda definir que a inspeção vem por último e leva três minutos. Para cada variável, exceto *inspeção,* adicionamos uma restrição da forma $X + d_X \leq$ Inspeção. Finalmente, suponha que haja uma exigência para ter todo o conjunto realizado em 30 minutos. Podemos conseguir isso limitando o domínio de todas as variáveis:

$$D_i = \{1, 2, 3, \ldots, 27\}.$$

Esse problema em particular é trivial para resolver, mas foram aplicados PSRs para agendamento de tarefas como esse, com milhares de variáveis. Em alguns casos, existem restrições complicadas que são difíceis de especificar no formalismo PSR, e técnicas mais avançadas de planejamento são utilizadas, como as que serão discutidas no Capítulo 11.

6.1.3 Variações do formalismo PSR

A espécie mais simples de PSR envolvendo variáveis tem **domínios discretos e finitos**. Os problemas de coloração de mapas e de agendamento de prazos são desse tipo. O problema de oito rainhas descrito no Capítulo 3 também pode ser visto como um PSR de domínios finitos, onde as variáveis $Q_1, \ldots, Q_8$ são as posições de cada rainha nas colunas

1, …, 8, e onde cada variável tem o domínio $D_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$.

Um domínio discreto pode ser **infinito**, como o conjunto de números inteiros ou de cadeias de caracteres (se não colocássemos um prazo para o problema de agendamento de tarefas, haveria um número infinito de horários de início para cada variável). No caso de domínios infinitos, não é mais possível descrever restrições enumerando todas as combinações de valores permitidas. Em vez disso, deve ser utilizada uma **linguagem de restrições** para compreender restrições diretamente, tais como $T_1 + d_1 \leq T_2$, sem enumerar o conjunto de pares de valores permitidos para (T_1, T_2).

Existem algoritmos de solução especial (que não discutiremos aqui) para **restrições lineares** sobre variáveis inteiras, ou seja, restrições como a que acabamos de mostrar, em que cada variável só aparece em forma linear. É possível mostrar que não existe nenhum algoritmo para resolver **restrições não lineares** sobre variáveis inteiras.

Os problemas de satisfação de restrições com **domínios contínuos** são muito comuns no mundo real e amplamente estudados no campo de pesquisa operacional. Por exemplo, o escalonamento de experimentos no telescópio espacial Hubble exige sincronização muito precisa de observações; o começo e o fim de cada observação e a manobra são variáveis de valores contínuos que devem obedecer a uma variedade de restrições astronômicas, de precedência e de energia. A categoria mais conhecida de PSRs de domínios contínuos é a dos problemas de **programação linear**, em que as restrições devem ser equações ou inequações lineares que formam uma região *convexa*. Os problemas de programação linear podem ser resolvidos em tempo polinomial no número de variáveis. Também foram estudados problemas com diferentes tipos de restrições e funções objetivo — programação quadrática, programação cônica de segunda ordem, e assim por diante.

Além de examinar os tipos de variáveis que podem aparecer em PSRs, é útil examinar os tipos de restrições. O tipo mais simples é a **restrição unária**, que restringe o valor de uma única variável. Por exemplo, no probema de coloração do mapa, os australianos do sul poderiam não tolerar a cor verde; isso pode ser expresso com a restrição unária $\langle (AM), AM \neq verde \rangle$.

Uma **restrição binária** relaciona duas variáveis. Por exemplo, $AM \neq NGS$ é uma restrição binária. Um PSR binário é aquele que só tem restrições binárias; ele pode ser representado como um grafo de restrições, conforme mostra a Figura 6.1(b).

Também podemos descrever restrições de ordem superior, tal como afirmar que o valor de *Y* está entre *X* e *Z*, com a restrição ternária *Entre*(*X*, *Y*, *Z*).

A restrição que envolve um número arbitrário de variáveis é chamada **de restrição global** (o nome é tradicional, mas confuso, porque não é necessário envolver todas as variáveis em um problema). Uma das restrições globais mais comuns é *TodosDiferentes*, que determina que todas as variáveis envolvidas em uma restrição devem ter valores diferentes. Em problemas de Sudoku (veja a Seção 6.2.6), todas as variáveis em uma linha ou coluna devem satisfazer uma restrição *TodosDiferentes*. Outro exemplo é fornecido pelos quebra-cabeças **criptoaritméticos** (veja a Figura 6.2(a)). Cada letra em um quebra-cabeça criptoaritmético representa um dígito diferente. No caso da Figura 6.2(a), isso seria representado como restrição global *TodosDiferentes*(*F*, *T*, *U*, *W*, *R*, *O*). As restrições de adição sobre as quatro colunas do quebra-cabeça também envolvem diversas variáveis e podem ser representadas como as seguintes restrições *n*-árias:

$$O + O = R + 10 \cdot C_{10}$$
$$C_{10} + W + W = U + 10 \cdot C_{100}$$
$$C_{100} + T + T = O + 10 \cdot C_{1000}$$
$$C_{1000} = F,$$

onde C_{10}, C_{100} e C_{1000} são **variáveis auxiliares** representando o dígito transportado para a coluna décima, centésima, milésima. Essas restrições de ordem alta podem ser representadas em um **hipergrafo de restrições**, como mostra a Figura 6.2(b). Um hipergrafo consiste em nós ordinários (os círculos na figura) e os hipernós (os quadrados), representam restrições *n*-árias.

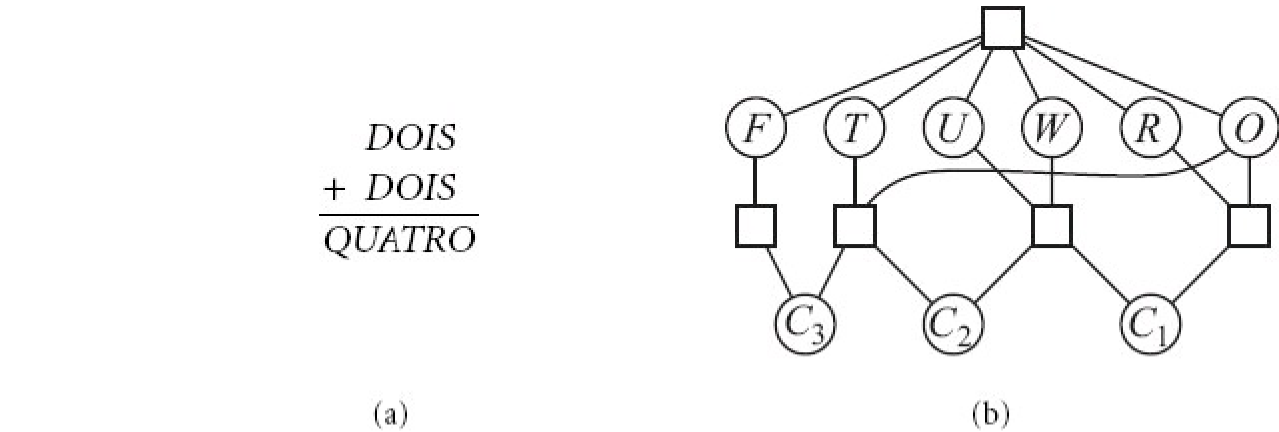


Figura 6.2 (a) Um problema criptoaritmético. Cada letra representa um dígito distinto; o objetivo é encontrar uma substituição de letras por dígitos, tal que a soma aritmética resultante seja correta, com a restrição adicional de não se permitir nenhum zero à esquerda. (b) O hipergrafo de restrições para o problema criptoaritmético, mostrando a restrição *TodosDiferentes* (caixa quadrada no topo), bem como as restrições de adição de colunas (as quatro caixas quadradas ao centro). As variáveis C_1, C_2 e C_3 representam os dígitos de transporte para as três colunas.

Alternativamente, como o Exercício 6.6 pede que você prove, cada restrição de domínio finito pode ser reduzida a um conjunto de restrições binárias se forem introduzidas variáveis auxiliares suficientes para que possamos transformar qualquer PSR em um com restrições apenas binárias, tornando os algoritmos mais simples. Outra forma de converter um PSR *n*-ário em um binário é pela transformação **grafo dual**: criar um novo grafo no qual haverá uma variável para cada restrição no

grafo original e uma restrição binária para cada par de restrições no grafo original que partilha as variáveis.

Por exemplo, se o grafo original tiver as variáveis $\{X, Y, Z\}$ e as restrições $\langle (X, Y, Z), C_1 \rangle$ e $\langle (X, Y), C_2 \rangle$, então o grafo dual teria as variáveis $\{C_1, C_2\}$ com a restrição binária $\langle (X, Y), R_1 \rangle$, onde (X, Y) são as variáveis compartilhadas e R_1 é uma nova relação que define a restrição entre as variáveis compartilhadas, conforme especificado pelo original C_1 e C_2.

No entanto, existem duas razões pelas quais é preferível uma restrição global, tal como *TodosDiferentes,* do que um conjunto de restrições binárias. Primeiro, é mais fácil e menos suscetível a erros escrever a descrição do problema usando *TodosDiferentes*. Segundo, é possível projetar algoritmos de inferências de propósitos especiais para restrições globais que não estejam disponíveis para um conjunto de restrições mais primitivas. Descreveremos esses algoritmos de inferência na Seção 6.2.5.

As restrições que descrevemos até agora foram todas restrições absolutas, cuja violação elimina uma solução potencial. Muitos PSRs reais incluem **restrições de preferência**, indicando as soluções preferidas. Por exemplo, em um problema de elaboração de horário de aula em uma universidade há restrições absolutas, como a que nenhum professor pode dar duas aulas ao mesmo tempo. Mas podemos também permitir restrições de preferência: o professor R pode preferir lecionar pela manhã, enquanto o professor N pode preferir lecionar à tarde. Um agendamento que tenha o professor R lecionando às 14 horas ainda seria uma solução (a menos que o professor R seja o chefe do departamento), mas não seria uma solução ótima. As restrições de preferência frequentemente podem ser codificadas como custos sobre atribuições de variáveis individuais — por exemplo, a atribuição de um horário de aula à tarde para o professor R custa dois pontos contra a função objetivo global, enquanto um horário pela manhã custa um. Com essa formulação, os PSRs com preferências podem ser resolvidos utilizando-se métodos de busca de otimização, baseados em caminhos ou locais. Chamamos tal problema de **problema de otimização de restrição**, ou POR. Problemas de programação linear realizam esse tipo de otimização.

6.2 PROPAGAÇÃO DE RESTRIÇÃO: INFERÊNCIA EM PSRs

Na busca regular em espaços de estados, um algoritmo pode fazer apenas uma coisa: busca. Em PSRs há uma escolha: um algoritmo pode buscar (escolher uma nova atribuição para a variável de várias possibilidades) ou fazer um tipo específico de **inferência** chamada **propagação de restrições**: utilizando as restrições para reduzir o número de valores válidos para uma variável, o que, por sua vez, pode reduzir os valores válidos para outra variável, e assim por diante. A propagação de restrição pode ser interligada com a busca ou pode ser feita como uma etapa de pré-processamento, antes que a busca seja iniciada. Às vezes, esse pré-processamento pode resolver todo o problema e, assim, não é requerida nenhuma busca.

A ideia-chave é a **consistência local**. Se tratarmos cada variável como um nó em um grafo (veja a Figura 6.1 (b)) e cada restrição binária como um arco, o processo de aplicar a consistência local em cada parte do grafo faz com que os valores inconsistentes sejam eliminados em todo o grafo. Existem

tipos diferentes de consistência local, que cobriremos agora.

6.2.1 Consistência de nó

A única variável (correspondente a um nó na rede PSR) é **nó-consistente** se todos os valores no domínio da variável satisfizerem as restrições unárias da variável. Por exemplo, na variante do problema de coloração do mapa da Austrália (Figura 6.1), onde os australianos do sul não gostam de verde, a variável AM começa com o domínio {*vermelho, verde, azul*}, e podemos torná-la nó-consistente, eliminando o *verde*, deixando *AM* com o domínio reduzido {*vermelho, azul*}. Dizemos que uma rede é nó-consistente se todas as variáveis da rede forem nó-consistentes.

É sempre possível eliminar todas as restrições unárias em um PSR executando nó-consistência. Também é possível transformar todas as restrições *n*-árias em binárias (veja o Exercício 6.6). Por isso, é comum definir solucionadores de PSR que trabalham apenas com as restrições binárias; faremos essa suposição no restante deste capítulo, exceto onde indicado.

6.2.2 Consistência de arco

Uma variável em um PSR é **arco-consistente** se todos os valores em seu domínio satisfizerem as restrições binárias da variável. Mais formalmente, X_i é arco-consistente com relação à outra variável X_j se separa cada valor no domínio atual D_i quando houver algum valor no domínio D_j que satisfaça a restrição binária sobre o arco (X_i, X_j). Uma rede é arco-consistente se cada variável for arco-consistente com todas as outras variáveis. Por exemplo, considere a restrição $Y = X^2$, onde o domínio de X e Y é o conjunto de dígitos. Podemos escrever essa restrição explicitamente como

$$\langle (X, Y), \{(0,0), (1,1), (2,4), (3,9))\} \rangle.$$

Para tornar X arco-consistente com relação a Y, reduzimos o domínio de X para {0, 1, 2, 3}. Se tornarmos também Y arco-consistente com relação a X, o domínio de Y torna-se {0, 1, 4, 9} e todo o PSR será arco-consistente.

Por outro lado, a consistência de arco não pode fazer nada com respeito ao problema de coloração do mapa da Austrália. Considere a seguinte restrição de desigualdade em (*AM*, *AO*):

$$\{(vermelho, verde), (vermelho, azul), (verde, vermelho), (verde, azul), (azul, vermelho), (azul, verde)\}.$$

Não importa o valor escolhido para *AM* (ou para *AO*), há um valor válido para a outra variável. Assim, a aplicação de consistência de arco não tem efeito sobre os domínios de qualquer variável.

O algoritmo mais popular para a consistência de arco é chamado de AC-3 (veja a Figura 6.3). Para tornar toda a variável arco-consistente, o algoritmo AC-3 mantém uma fila de arcos a considerar (na verdade, a ordem de consideração não é importante, por isso a estrutura de dados é realmente um conjunto, mas a tradição a considera uma fila). Inicialmente, a fila contém todos os arcos no PSR. O AC-3, então, saltará de um arco arbitrário (X_i, X_j) da fila e se tornará X_i arco-

consistente com relação a X_j. Se deixar D_i inalterado, o algoritmo apenas se moverá para o arco seguinte. Mas, se isso modificar D_i (torna o domínio menor), adicionamos todos os arcos à fila (X_k, X_i), onde X_k é um vizinho de X_i. Precisamos fazer isso porque a alteração em D_i pode permitir novas reduções nos domínios de D_k, mesmo que tenhamos considerado anteriormente X_k. Se D_i for modificado para nada, saberemos que todo o PSR não tem uma solução consistente e o AC-3 poderá retornar falha imediatamente. Caso contrário, continuamos a verificar, tentando remover os valores dos domínios das variáveis até que não haja mais arcos na fila. Nesse ponto, estaremos com um PSR que é equivalente ao PSR original — ambos têm as mesmas soluções, mas, na maioria dos casos, o PSR arco-consistente será mais rápido na busca porque suas variáveis têm domínios menores.

função AC-3(*psr*) **retornar** falso se uma inconsistência for encontrada e verdadeiro se não for
  entradas: *psr*, um PSR binário com componentes (X, D, C)
  variáveis locais: *fila*, uma fila de arcos, inicialmente todos os arcos em *psr*

  enquanto *fila* não está vazia **faça**
    (X_i, X_j) ← REMOVAR-PRIMEIRO(*fila*)
    Se REVISAR(*psr*, X_i, X_j) **então**
      Se tamanho de $D_i = 0$ **então retornar** *falso*
      para cada X_k **em** X_i.VIZINHANÇAS - {X_j} **faça**
        adicionar (X_k, X_i) à fila
  retornar verdadeiro

---

função REVISAR(*psr*, X_i, X_j) **retornar** verdadeiro se revisarmos o domínio de *Xi*
  revisado ← *falso*
  para cada *x* **em** D_i **faça**
    se nenhum valor *y* em D_j permite (*x*,*y*) para satisfazer a restrição entre X_i e X_j **então**
      excluir *x* de D_i
      revisado ← *verdadeiro*
  retornar *revisado*

Figura 6.3 Algoritmo de consistência de arco AC-3. Após aplicar AC-3, cada arco fica arco-consistente ou alguma variável tem um domínio vazio, indicando que o PSR não pode ser resolvido. O nome “AC-3” foi utilizado pelo inventor do algoritmo (Mackworth, 1977) por ser a terceira versão desenvolvida no documento.

A complexidade de AC-3 pode ser analisada como segue. Assumir um PSR com *n* variáveis, cada uma com o tamanho de domínio, no máximo, *d*, e com *c* restrições binárias (arcos). Cada arco (X_k, X_i) pode ser inserido na fila apenas d vezes porque X_i tem, no máximo, *d* valores para excluir. A verificação da consistência de um arco pode ser feita no tempo $O(d^2)$, então temos $O(cd^3)$ total de casos de pior tempo.[1]

É possível estender a noção de consistência de arco para manusear restrições n-árias em vez de apenas binárias; isso se chama **consistência de arco generalizada** ou, algumas vezes, consistência de hiperarco, dependendo do autor. Uma variável X_i é **arco-consistente generalizada** com relação a uma restrição n-ária se, para cada valor v no domínio X_i, existir uma dupla de valores que seja um membro da restrição, tenha todos os seus valores retirados dos domínios das variáveis correspondentes e tenha seu componente X_i igual a v. Por exemplo, se todas as variáveis tiverem o domínio {0, 1, 2, 3}, então para tornar a variável X consistente com a restrição $X < Y < Z$, teríamos que eliminar 2 e 3 do domínio de X porque a restrição não pode ser satisfeita quando X é 2 ou 3.

6.2.3 Consistência de caminho

A consistência de arco pode contribuir muito para reduzir os domínios das variáveis, por vezes encontrar uma solução (ao reduzir cada domínio ao tamanho 1) e, por vezes, constatar que o PSR não pode ser resolvido (através da redução de algum domínio ao tamanho 0). Mas, para outras redes, a consistência de arco não faz inferências suficientes. Considere o problema de coloração do mapa da Austrália, mas apenas com duas cores permitidas, vermelho e azul. A consistência de arco não pode fazer nada porque todas as variáveis já são arco-consistentes: cada uma pode estar do outro lado do arco vermelho e azul (ou vice-versa). Mas certamente não há solução para o problema porque, devido ao oeste da Austrália, e os territórios do norte e do sul tocarem um no outro, precisamos de pelo menos três cores só para eles.

A consistência de arco fixa os domínios para baixo (restrições unárias) utilizando os arcos (restrições binárias). Para progredir com problemas como de coloração de mapa, precisamos de uma noção forte de consistência. A c**onsistência de caminho** fixa as restrições binárias utilizando restrições implícitas que são inferidas pela verificação do triplo de variáveis.

Um conjunto de duas variáveis $\{X_i, X_j\}$ é consistente de caminho em relação a uma terceira variável X_m se, para cada atribuição $\{X_i = a, X_j = b\}$ consistente com as restrições em $\{Xi, Xj\}$, houver uma atribuição para X_m que satisfaça as restrições em $\{X_i, X_m\}$ e $\{X_m, X_j\}$. Isso se chama consistência de caminho porque se pode cogitar isso ao olhar para um caminho de X_i para X_j com X_m ao meio.

Vejamos como a consistência de caminho se sai ao colorir o mapa da Austrália com duas cores. Faremos o conjunto {*AO*, *AM*} consistente de caminho em relação a *TN*. Começaremos por enumerar as atribuições consistentes para o conjunto. Nesse caso, há apenas duas: {*AO* = *vermelho*, *AM* = *azul*}e {*AO* = *azul*, *AM* = *vermelho*}. Podemos ver que nas duas atribuições *TN* não pode ser nem vermelho nem azul (porque entraria em conflito com *AO* ou *AM*). Por não haver uma opção válida para *TN*, eliminaremos as duas atribuições e finalizaremos sem atribuição válida para {*AO*, *AM*}. Portanto, sabemos que não pode haver solução para esse problema. O algoritmo PC-2 (Mackworth, 1977) alcança a consistência de caminho da mesma maneira que o AC-3 alcança a consistência de arco. Por ser tão semelhante, não o demonstraremos aqui.

6.2.4 *K*-Consistência

Com a noção de *k*-**consistência** pode-se definir formas mais fortes de propagação. A PSR é k-consistente se, para qualquer conjunto de $k-1$ variáveis e para qualquer atribuição consistente para aquelas variáveis, um valor consistente puder sempre ser atribuído a qualquer variável *k*-ésima. 1-consistência determina que, dado o conjunto vazio, podemos tornar qualquer conjunto de uma variável consistente: isso é o que se chama consistência de nó. 2-consistência é a mesma consistência de arco. Para redes de restrição binária, 3-consistência é a mesma consistência de caminho.

A PSR é **fortemente** *k*-**consistente** se for *k*-consistente e também $(k-1)$-consistente, $(k-2)$-consistente, … até 1-consistente. Agora, suponha que tenhamos um PSR com *n* nós e o tornamos fortemente *n*-consistente (ou seja, fortemente *k*-consistente para $k = n$). Podemos, então, resolver o problema da seguinte maneira: primeiro, escolhemos um valor consistente para X_1. Estamos então garantidos de ser capazes de escolher um valor para X_2 porque o grafo é 2-consistente, para X_3 porque é 3-consistente, e assim por diante. Para cada variável X_i, precisamos apenas buscar através de *d* valores no domínio para encontrar um valor consistente com $X_1, \ldots, X_{i-1}$. Temos a garantia de encontrar uma solução no tempo $O(n^2d)$. Claro, não há almoço grátis: para qualquer algoritmo estabelecer a consistência, *n* terá que levar *n* tempo exponencial no pior caso. Pior, a consistência *n* também requer espaço que é exponencial em *n*. O problema de memória é ainda mais grave do que o de tempo. Na prática, determinar o nível adequado de verificação de consistência, sobretudo, é uma ciência empírica. Pode-se dizer que praticantes calculam a 2-consistência comumente, e menos comumente a 3-consistência.

6.2.5 Restrições globais

Lembre-se de que uma **restrição global** é aquela que envolve um número arbitrário de variáveis (mas não necessariamente todas as variáveis). Em problemas reais ocorrem com frequência restrições globais e podem ser tratadas por algoritmos com propósitos especiais que são mais eficientes que os métodos de uso geral descritos até agora. Por exemplo, a restrição *TodosDiferentes* determina que todas as variáveis envolvidas devem ter valores distintos (como no problema criptoaritmético anterior e nos quebra-cabeças Sudoku adiante). Uma forma simples de detecção de inconsistência para as restrições *TodosDiferentes* funciona da seguinte forma: se *m* variáveis estiverem envolvidas na restrição e se tiverem *n* possíveis valores completamente distintos, e $m > n$, então a restrição não poderá ser satisfeita.

Isso conduz ao seguinte algoritmo simples: primeiro, retire qualquer variável da restrição que tenha um domínio avulso e exclua o valor dessa variável dos domínios das variáveis restantes. Repita enquanto existirem variáveis avulsas. Se em algum momento for produzido um domínio vazio ou restarem mais variáveis do que valores de domínio, foi detectada uma inconsistência.

Esse método pode detectar a inconsistência na atribuição $\{AO = vermelho, NGS = vermelho\}$ para a Figura 6.1. Observe que as variáveis *AM*, *TN* e *Q* estão efetivamente ligadas por uma restrição *TodosDiferentes* porque cada par deve ter duas cores diferentes. Após aplicar AC-3 com a restrição

parcial, o domínio de cada variável será reduzido para {*verde, azul*}. Isto é, temos três variáveis e apenas duas cores, por isso a restrição *TodosDiferentes* foi violada. Assim, um procedimento simples de consistência de uma restrição de ordem superior é muitas vezes mais eficaz do que aplicar a consistência de arco para um conjunto equivalente de restrições binárias. Existem mais algoritmos de inferência complexos para *TodosDiferentes* (veja Van Hoeve e Katriel, 2006) que propagam mais restrições, mas computacionalmente a execução é mais cara.

Outra restrição importante de ordem superior é a **restrição de recurso**, chamada algumas vezes de restrição *atmost*. Por exemplo, em um problema de programação, façamos P_1, …, P_4 denotar o número de pessoal atribuído a cada uma de quatro tarefas. A restrição de que não mais que 10 pessoas sejam designadas no total é escrita como *atmost* (10, P_1, P_2, P_3, P_4). Podemos detectar uma inconsistência simplesmente verificando a soma dos valores mínimos dos domínios atuais; por exemplo, se cada variável tiver um domínio {3, 4, 5, 6}, a restrição *atmost* não poderá ser satisfeita. Podemos também garantir a consistência, excluindo o valor máximo de qualquer domínio se não for consistente com os valores mínimos dos outros domínios. Assim, se cada variável em nosso exemplo tiver o domínio {2, 3, 4, 5, 6}, os valores 5 e 6 poderão ser excluídos de cada domínio.

Para grandes problemas de recursos limitados a valores inteiros, tais como os problemas de logística envolvendo a movimentação de milhares de pessoas em centenas de veículos, geralmente não é possível representar o domínio de cada variável como um grande conjunto de números inteiros e gradualmente reduzir esse conjunto com métodos de verificação de consistência. Em vez disso, os domínios são representados por limites superiores e inferiores, e são geridos por **propagação de limites**. Por exemplo, em um problema de programação de voo, vamos supor que existam dois voos, F_1 e F_2, para os quais a capacidade dos aviões é de 165 e 385, respectivamente. Os domínios iniciais para o número de passageiros em cada voo são, então,

$$D_1 = [0, 165] \text{ e } D_2 = [0, 385].$$

Agora, suponha que tenhamos a restrição adicional de que dois voos juntos devem levar 420 pessoas: $F_1 + F_2 = 420$. Propagando as restrições de limites, reduzimos os domínios para

$$D_1 = [35, 165] \text{ e } D_2 = [255, 385].$$

Dizemos que um PSR é **de limites consistentes** se, para cada variável *X*, e tanto para os valores de *X* do limite superior como para do inferior, existir algum valor de *Y* que satisfaça a restrição entre *X* e *Y* para cada variável Y. Esse tipo de propagação de limite é amplamente utilizado em problemas práticos de restrição.

6.2.6 Exemplo: Sudoku

O quebra-cabeça popular **Sudoku** apresentou a milhões de pessoas os problemas de satisfação de restrição, apesar de talvez poderem não reconhecê-los. Uma tábua de Sudoku é composta de 81 quadrados, alguns dos quais são preenchidos inicialmente com os dígitos e 1 a 9. O enigma é

preencher todos os quadrados restantes de tal forma que nenhum dígito apareça duas vezes em qualquer linha, coluna ou caixa 3 × 3 (ver a Figura 6.4). A linha, coluna ou caixa é chamada de **unidade**.

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Figura 6.4 (a) Um quebra-cabeça Sudoku e (b) sua solução.

A propriedade dos quebra-cabeças de Sudoku que são impressos em jornais e livros de quebra-cabeça é que existe exatamente apenas uma solução. Apesar de alguns serem complicados de resolver à mão, levando dezenas de minutos, até mesmo os problemas mais difíceis de Sudoku podem ser resolvidos por um solucionador PSR em menos de 0,1 segundo.

Um quebra-cabeça Sudoku pode ser considerado um PSR com 81 variáveis, uma para cada quadrado. Utilizamos os nomes das variáveis *A1* até *A9* para a linha de cima (da esquerda para a direita), para a linha de baixo de *I1* até *I9*. Os quadrados vazios têm o domínio {1, 2, 3, 4, 5, 6, 7, 8, 9} e os quadrados pré-preenchidos têm um domínio que consiste em um único valor. Além disso, existem 27 restrições *TodasDiferentes*: uma para cada linha, coluna e caixa de nove quadrados.

Restrições *TodosDiferentes*: um para cada linha, coluna e caixa com 9 quadrados.

$$
\begin{array}{l}
TodosDiferentes(A1, A2, A3, A4, A5, A6, A7, A8, A9) \\
TodosDiferentes(B1, B2, B3, B4, B5, B6, B7, B8, B9) \\
\ldots \\
TodosDiferentes(A1, B1, C1, D1, E1, F1, G1, H1, I1) \\
TodosDiferentes(A2, B2, C2, D2, E2, F2, G2, H2, I2) \\
\ldots \\
TodosDiferentes(A1, A2, A3, B1, B2, B3, C1, C2, C3) \\
TodosDiferentes(A4, A5, A6, B4, B5, B6, C4, C5, C6) \\
\ldots
\end{array}
$$

Vamos ver até onde a consistência de arco pode nos levar. Assuma que as restrições *TodosDiferentes* foram expandidas em restrições binárias (tais como A1 ≠ *A2*) para que possamos aplicar o algoritmo AC-3 diretamente. Considere a variável *E6* da Figura 6.4 (a) — o quadrado vazio entre o 2 e o 8 na caixa do meio. Podemos remover das restrições na caixa, não apenas 2 e 8, mas também 1 e 7 do domínio *E6*. Das restrições nessa coluna, podemos eliminar 5, 6, 2, 8, 9 e 3.

Deixando *E6* com um domínio de {4}; em outras palavras, sabemos a resposta para *E6*. Agora considere a variável *I6* — o quadrado na caixa do meio embaixo cercado por 1, 3 e 3. Aplicando a consistência de arco em sua coluna, eliminamos 5, 6, 2, 4 (já que sabemos agora que *E6* deve ser 4), 8, 9 e 3. Eliminamos 1 por consistência de arco com *I5*, e restou apenas o valor 7 no domínio de *I6*. Agora existem oito valores conhecidos na coluna 6, de modo que a consistência de arco pode inferir que *A6* deve ser 1. A inferência continua ao longo dessas linhas e, eventualmente, AC-3 pode resolver todo o quebra-cabeça — todas as variáveis têm os seus domínios reduzidos a um único valor, como mostrado na Figura 6.4 (b).

Certamente o Sudoku perderia o seu apelo se cada quebra-cabeça puder ser resolvido por uma aplicação mecânica de AC-3, e realmente o AC-3 só funciona para os quebra-cabeças de Sudoku mais fáceis. Aqueles um pouco mais difíceis podem ser resolvidos por PC-2, mas a um custo computacional maior: há 255.960 restrições diferentes de caminho para considerar em um quebra-cabeça Sudoku. Para resolver os quebra-cabeças mais difíceis e fazer progressos eficientes, teremos que ser mais espertos.

De fato, para o solucionador humano, o apelo do quebra-cabeças Sudoku é a necessidade de ser criativo na aplicação de estratégias de inferência mais complexas. Os aficionados lhes dão nomes coloridos, tais como triplos nus". Essa estratégia funciona da seguinte maneira: em qualquer unidade (linha, coluna ou caixa), encontre três quadrados em que cada um tenha um domínio que contenha os mesmos três números ou um subconjunto desses números. Por exemplo, os três domínios podem ser {1, 8}, {3, 8} e {1, 3, 8}. A partir deles não sabemos qual quadrado contém 1, 3 ou 8, mas sabemos que os três números deverão ser distribuídos entre os três quadrados. Portanto, podemos remover 1, 3 e 8 dos domínios de todos os *outros* quadrados na unidade.

É interessante observar como se pode ir longe sem dizer muito que seja específico ao Sudoku. Naturalmente temos que dizer que existem 81 variáveis, que os seus domínios são os dígitos 1 a 9 e que existem 27 restrições *TodosDiferentes*. Mas, além disso, todas as estratégias — consistência de arco, consistência de caminho etc., geralmente são aplicáveis a todos os PSRs, não apenas para os problemas de Sudoku. Mesmo os triplos nus são realmente uma estratégia para reforçar a consistência das restrições de *TodosDiferentes* e não tem nada a ver com o Sudoku *per se*. Esse é o poder do formalismo PSR: para cada nova área problemática, precisamos apenas definir o problema em termos de restrições; em seguida os mecanismos gerais de resolução de restrição podem assumir.

6.3 BUSCA COM RETROCESSO PARA PSRs

Os problemas de Sudoku são projetados para ser resolvidos por inferência sobre as restrições. Mas muitos outros PSRs não podem ser resolvidos apenas por inferência; chega um momento em que devemos procurar uma solução. Nesta seção, veremos os algoritmos de busca com retrocesso que funcionam com atribuições parciais; na próxima seção veremos algoritmos de busca local sobre tarefas completas.

Poderíamos aplicar uma busca em profundidade limitada padrão (do Capítulo 3). Um estado seria uma atribuição parcial, e uma ação seria a adição *var* = *valor* para a atribuição. Mas, para um PSR com *n* variáveis de tamanho de domínio *d*, logo notamos algo terrível: o fator de ramificação no nível

superior é nd porque qualquer valor entre d valores pode ser atribuído a qualquer das n variáveis. No próximo nível, o fator de ramificação é $(n - 1)d$, e assim por diante para n níveis. Geramos uma árvore com $n! \cdot d^n$ folhas, embora existam apenas d^n atribuições completas possíveis!

Nossa formulação de problema aparentemente razoável mas ingênua ignorou uma propriedade crucial, comum a todo os PSRs: a **comutatividade**. Um problema é comutativo se a ordem de aplicação de qualquer conjunto de ações dado não tem nenhum efeito sobre o resultado. PSRs são comutativos, pois ao atribuir valores às variáveis, chegamos à mesma tarefa parcial, independentemente da ordem. Portanto, basta considerarmos uma *única* variável em cada nó na árvore de busca.

Por exemplo, no nó raiz de uma árvore de busca para coloração do mapa da Austrália, poderíamos ter uma escolha entre *AM* = *vermelho*, *AM* = *verde* e *AM* = *azul*, mas nunca escolheríamos entre *AM* = *vermelho* e *AO* = *azul*. Com essa restrição, o número de folhas é d^n, como seria de esperar.

A expressão **busca com retrocesso** é utilizada para indicar uma busca em profundidade que escolhe valores para uma variável de cada vez e que efetua o retrocesso quando uma variável não tem valores válidos restantes a serem atribuídos. O algoritmo é mostrado na Figura 6.5. Ele escolhe repetidamente uma variável não atribuída e depois experimenta todos os valores no domínio da variável, por vez, tentando encontrar uma solução. Se for detectada uma inconsistência, o RETROCESSO-RECURSSIVO retorna falha, fazendo com que a chamada anterior tente outro valor. Parte da árvore de busca para o problema da Austrália é mostrada na Figura 6.6, onde atribuímos variáveis na ordem *AO*, *TN*, *Q*,… Como a representação de PSRs é padronizada, não há necessidade de prover PESQUISA-COM-RETROCESSO com um estado inicial específico de domínio, função ação, modelo de transição ou teste de objetivo.

```
função PESQUISA-COM-RETROCESSO(psr) retorna uma solução ou falha
  retornar RETROCESSO-RECURSIVO({ }, psr)

função RETROCESSO-RECURSIVO(atribuição, psr) retorna uma solução ou falha
  se atribuição é completa então retornar atribuição
  var ← SELECIONAR-VARIÁVEL-NÃO-ATRIBUÍDA(psr)
  para cada valor em VALORES-DE-ORDEM-NO-DOMÍNIO(var, atribuição, psr) faça
    se valor é consistente com atribuição então
      adicionar { var = valor } a atribuição
      inferência ← INFERÊNCIA(psr, atribuição, var)
      se inferência ≠ falhar então
        adicione inferência para atribuição
        resultado ← RETROCESSO-RECURSIVO(atribuição, psr)
        se resultado ≠ falha então
          retornar resultado
    remover { var = valor } de atribuição
  retornar falha
```

Figura 6.5 Um algoritmo simples com retrocesso para problemas de satisfação de restrições. O

algoritmo é modelado sobre a busca recursiva em profundidade do Capítulo 3. Pela variação das funções SELECIONAR-VARIÁVEL-NÃO-ATRIBUÍDA e VALORES-DE-ORDEM-NO-DOMÍNIO podemos implementar as heurísticas de uso geral descritas no texto. A função INFERÊNCIA pode ser utilizada opcionalmente para impor arco-, caminho- ou *k*-consistência, conforme desejado. Se uma escolha de valor levar ao fracasso (por INFERÊNCIA ou por RETROCESSO-RECURSSIVO), as atribuições de valor (incluindo as realizadas por INFERÊNCIA) são removidas da atribuição e um novo valor será experimentado.

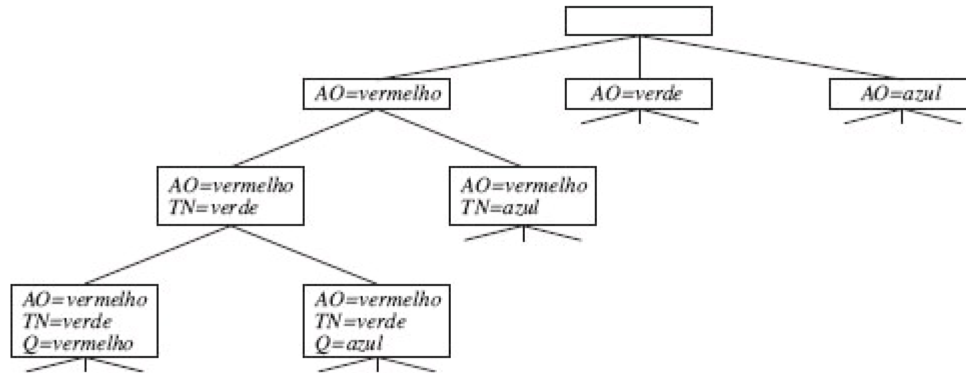


Figura 6.6 Parte da árvore de busca gerada por retrocesso simples para o problema de coloração de mapa da Figura 6.1.

Note que a PESQUISA-COM-RETROCESSO mantém apenas uma representação única de um estado e altera essa representação, em vez de criar novas, conforme descrito anteriormente.

No Capítulo 3 melhoramos o fraco desempenho de algoritmos de busca sem informações fornecendo a eles funções heurísticas específicas do domínio, derivadas de nosso conhecimento do problema. Ocorre que podemos resolver PSRs de maneira eficiente sem tal conhecimento específico de domínio. Em vez disso, podemos acrescentar um pouco de sofisticação para as funções não especificadas na Figura 6.5 utilizando-as para abordar as seguintes questões:

1. Que variável deve ser atribuída em seguida (SELECIONAR-VARIÁVEL-NÃO-ATRIBUÍDA), e em que ordem seus valores devem ser experimentados (VALORES-DE-ORDEM-NO-DOMÍNIO)?
2. Que inferências devem ser realizadas a cada etapa na busca (INFERÊNCIA)?
3. Quando a pesquisa chega a uma atribuição que viola uma restrição, a busca pode evitar a repetição desse fracasso?

As subseções a seguir respondem a cada uma dessas perguntas.

6.3.1 Ordenação de variáveis e valores

O algoritmo com retrocesso contém a linha:

var ← SELECIONAR-VARIÁVEL-NÃO-ATRIBUÍDA-VARIÁVEIS(*psr*)

A estratégia mais simples para SELECIONAR-VARIÁVEL-NÃO-ATRIBUÍDA é escolher a próxima variável não atribuída na ordem $\{X_1, X_2, \ldots\}$. Essa ordenação de variáveis estáticas raramente resulta na busca mais eficiente. Por exemplo, depois das atribuições de *AO* = *vermelho* e *TN* = *verde* na Figura 6.6, só existe um valor possível para *AM*; então, faz sentido atribuir em seguida *AM* = *azul* em lugar de atribuir *Q*. De fato, depois de *AM* ser atribuído, as escolhas para *Q*, *NGS* e *V* são todas forçadas. Essa ideia intuitiva — escolher a variável com o menor número de valores "válidos" — é chamada heurística de **valores restantes mínimos** (VRM). Ela também é chamada heurística de "variável mais restrita" ou de "primeira falha"; este último nome é dado porque ela escolhe uma variável que tem a maior probabilidade de provocar uma falha em breve, podando assim a árvore de busca. Se alguma variável *X* não tem mais valores válidos restantes, a heurística VRM selecionará *X* e a falha será detectada de imediato — evitando buscas inúteis por outras variáveis. A heurística VRM geralmente tem desempenho melhor do que uma ordenação ao acaso ou estática, às vezes por um fator de 1.000 ou mais, embora os resultados variem amplamente dependendo do problema.

A heurística de VRM não ajuda de modo algum na escolha da primeira região a colorir na Austrália porque inicialmente toda região tem três cores válidas. Nesse caso, a **heurística de grau** se mostra prática. Ela tenta reduzir o fator de ramificação em escolhas futuras selecionando a variável envolvida no maior número de restrições sobre outras variáveis não atribuídas. Na Figura 6.1, *AM* é a variável com grau mais alto, 5; as outras variáveis têm grau 2 ou 3, com exceção de *T*, que tem grau 0. De fato, uma vez que *AM* é escolhida, a aplicação da heurística de grau resolve o problema sem quaisquer etapas falsas — é possível escolher qualquer cor consistente em cada ponto de escolha e ainda chegar a uma solução sem retrocesso. A heurística de valores restantes mínimos normalmente é um guia mais poderoso, mas a heurística de grau pode ser útil como critério de desempate.

Uma vez que uma variável foi selecionada, o algoritmo deve decidir-se pela ordem em que seus valores devem ser examinados. Para isso, a heurística de **valor menos restritivo** pode ser efetiva em alguns casos. Ela prefere o valor que elimina o menor número possível de escolhas para as variáveis vizinhas no grafo de restrições. Por exemplo, suponha que na Figura 6.1 tenhamos gerado a atribuição parcial com *AO* = *vermelho* e *TN* = *verde* e que nossa próxima escolha seja relativa a *Q*. Azul seria uma escolha ruim porque elimina o último valor válido restante para *AM*, vizinho de *Q*. Portanto, a heurística de valor menos restritivo prefere vermelho a azul. Em geral, a heurística está tentando deixar a máxima flexibilidade para atribuições de variáveis subsequentes. É claro que, se estivermos tentando encontrar todas as soluções para um problema, não apenas a primeira, a ordenação não importará, porque, de qualquer maneira, teremos de considerar todos os valores. O mesmo é válido se não houver nenhuma solução para o problema.

Por que a seleção de variáveis deveria ser de "primeira falha", mas a seleção de valor de "última falha"? Parece que, para uma ampla variedade de problemas, uma ordenação de variável que escolhe uma variável com o número mínimo de valores restantes ajuda a minimizar o número de nós na árvore de busca através da poda precoce de grandes partes da árvore. Para a ordenação de valores, o truque é que precisamos apenas de uma solução, por isso faz sentido procurar pelos valores mais prováveis primeiro. Se quiséssemos enumerar todas as soluções em vez de apenas encontrar uma, a ordenação de valores seria irrelevante.

6.3.2 Intercalação de baixa inferência

Até agora vimos como o AC-3 e outros algoritmos podem inferir reduções no domínio de variáveis *antes* de iniciarmos a busca. Mas a inferência pode ser ainda mais poderosa no curso de uma busca: cada vez que fazemos uma escolha de um valor para uma variável, temos uma nova oportunidade de inferir reduções no domínio novo sobre as variáveis vizinhas.

Uma das formas mais simples de inferência é chamada de **verificação à frente**. Sempre que uma variável *X* é atribuída, o processo de verificação à frente estabelece a consistência de arco para ela: para cada variável não atribuída *Y* que está conectada por uma restrição a *X*, exclui do domínio de *Y* qualquer valor que esteja inconsistente com o valor escolhido para *X*. Devido à verificação à frente só fazer inferências de consistência de arco, não há razão para realizar a verificação à frente se a consistência de arco já foi realizada como uma etapa de pré-processamento.

A Figura 6.7 mostra o progresso da busca com retrocesso no PSR da Austrália com verificação à frente. Existem dois pontos importantes a observar sobre esse exemplo. Primeiro, note que, depois de se atribuir *AO* = *vermelho* e *Q* = *verde*, os domínios de *TN* e *AM* são reduzidos a um único valor; eliminamos por completo a ramificação nessas variáveis propagando informações a partir de *AO* e *Q*. Um segundo ponto a observar é que, depois de *V* = *azul*, o domínio de *AM* está vazio. Consequentemente, a verificação prévia detectou que a atribuição parcial {*AO* = *vermelho*, *Q* = *verde*, *V* = *azul*} é inconsistente com as restrições do problema e, assim, o algoritmo regressará imediatamente.

	AO	*TN*	*Q*	*NGS*	*V*	*AM*	*T*
Domínios iniciais	Vm Vd A	Vm Vd A	Vm Vd A	Vm Vd A	Vm Vd A	Vm Vd A	Vm Vd A
Depois de *AO=vermelho*	Vm	Vd A	Vm Vd A	Vm Vd A	Vm Vd A	Vd A	Vm Vd A
Depois de *Q=verde*	Vm	A	Vd	Vm A	Vm Vd A	A	Vm Vd A
Depois de *V=azul*	Vm	A	Vd	Vm	A		Vm Vd A

Figura 6.7 O progresso de uma busca de coloração de mapa com verificação à frente. *AO* = *vermelho* é atribuído primeiro; em seguida, a verificação à frente exclui *vermelho* dos domínios das variáveis vizinhas *TN* e *AM*. Depois de *Q* = *verde* é atribuído, *verde* é excluído dos domínios de *TN*, *AM* e *NGS*. Depois de *V* = *azul* é atribuído, *azul* é excluído dos domínios de *NGS* e *AM*, deixando *AM* sem valores válidos.

Para muitos problemas, a busca será mais eficiente se combinarmos a heurística VRM com a verificação à frente. Considere a Figura 6.7 após a atribuição de {*AO* = *vermelho*}. Intuitivamente, parece que essa atribuição restringe os seus vizinhos, *TN* e *AM*, de modo que devemos tratar as próximas variáveis e, depois, todas as outras variáveis vão pender para o lugar. Isso é exatamente o que acontece com VRM: *TN* e *AM* têm dois valores, então um deles é escolhido em primeiro lugar, depois o outro, então *Q*, *NGS* e *V* em ordem. Finalmente, *T* ainda tem três valores e qualquer um deles funciona. Podemos visualizar a verificação à frente como uma maneira eficiente de calcular a informação de forma incremental, o que a heurística VRM precisa para fazer seu trabalho.

Embora a verificação à frente detecte muitas inconsistências, não detecta todas elas. O problema é que ela torna a variável atual consistente de arco, mas não olha adiante para tornar todas as outras variáveis consistentes de arco. Por exemplo, considere a terceira linha da Figura 6.7. Ela mostra que,

quando *AO* é *vermelho* e *Q* é *verde*, *TN* e *AS* são forçados a ser azuis. A verificação à frente não olha adiante o suficiente para perceber que essa é uma inconsistência: *TN* e *AS* são adjacentes e por isso não podem ter o mesmo valor.

O algoritmo chamado MCA para **manutenção da consistência de arcos** (**MCA**) detecta essa inconsistência. Após atribuir um valor para a variável X_i, o procedimento INFERÊNCIA chama AC-3, mas em vez de uma fila de todos os arcos no PSR, começamos apenas com os arcos (X_j, X_i) para todos os X_j que são variáveis não atribuídas vizinhas de X_i. Desse ponto, o AC-3 faz a propagação de restrição da forma habitual e, se qualquer variável tiver o seu domínio reduzido para o conjunto vazio, falha a chamada para AC-3 e sabemos recuar imediatamente. Verificamos que o MCA é estritamente mais poderoso do que a verificação à frente porque a verificação à frente realiza a mesma coisa que o MCA sobre os arcos iniciais na fila do MCA, mas, ao contrário do MCA, a verificação à frente não propaga restrições recursivamente quando são feitas alterações nos domínios das variáveis.

6.3.3 Retrocesso inteligente: olhando para trás

O algoritmo PESQUISA-COM-RETROCESSO da Figura 6.5 tem uma norma muito simples sobre o que fazer quando um ramo da busca falha: voltar até a variável precedente e experimentar um valor diferente para ela. Isso se chama **retrocesso cronológico** porque o ponto de decisão *mais recente* é revisto. Nesta subseção, consideraremos melhores possibilidades.

Considere o que acontece quando aplicamos o retrocesso simples da Figura 6.1 com uma ordenação fixa de variáveis *Q*, *NGS*, *V*, *T*, *AM*, *AO*, *TN*. Vamos supor que geramos a atribuição parcial {*Q* = *vermelho*, *NGS* = *verde*, *V* = *azul*, *T* = *vermelho*}. Quando experimentarmos a próxima variável, *AM*, veremos que todo valor viola uma restrição. Voltamos até *T* e experimentamos uma nova cor para Tasmânia! Obviamente, isso é tolice — a mudança da cor atribuída à Tasmânia não pode solucionar o problema da Austrália meridional.

Uma abordagem mais inteligente para retorno é retroceder até uma variável que possa corrigir o problema, uma variável que foi responsável por tornar um dos valores possíveis de *AM* impossível. Para fazer isso, vamos acompanhar um conjunto de atribuições que estão em conflito com algum valor de *AM*. O conjunto (nesse caso, {*Q* = *vermelho*, *NGS* = *verde*, *V* = *azul*,}), é chamado **conjunto de conflito** para *AM*. O método de **retorno** regressa até a atribuição *mais recente* no conjunto de conflito; nesse caso, o retorno saltaria sobre a Tasmânia e experimentaria um novo valor para *V*. Esse método é facilmente implementado pela modificação para RETROCESSO-RECURSSIVO de forma que acumule o conjunto de conflito, ao mesmo tempo que verifica a atribuição de um valor válido. Se não for encontrado nenhum valor válido, o algoritmo deve retornar o elemento mais recente do conjunto de conflito, juntamente com o indicador de falha.

O leitor atento terá notado que a verificação à frente pode fornecer o conjunto de conflito sem trabalho extra: sempre que a verificação à frente baseada em uma atribuição $X = x$ exclui um valor do domínio de *Y*, ela deve adicionar $X = x$ ao conjunto de conflito de *Y*. Se o último valor for eliminado do domínio de *Y*, as atribuições no conjunto de conflito de *Y* serão adicionadas ao conjunto de

conflito de *X*. Então, quando chegarmos a *Y*, saberemos de imediato para onde regressar, se necessário.

O leitor mais atento terá notado algo estranho: o retorno ocorre quando todo valor em um domínio está em conflito com a atribuição corrente; no entanto, a verificação à frente detecta esse evento e impede que a busca alcance tal nó! De fato, é possível mostrar que toda ramificação podada por retorno também é podada por verificação à frente. Consequentemente, o retorno simples é redundante em uma busca de verificação à frente ou, na verdade, em uma busca que utilize verificação de consistência mais forte, como MAC.

Apesar das observações do parágrafo anterior, a ideia que rege o retorno continua a ser boa: efetuar o retrocesso com base nas razões da falha. O retorno nota a falha quando o domínio de uma variável se torna vazio, mas, em muitos casos, uma ramificação está condenada muito antes de acontecer isso. Considere mais uma vez a atribuição parcial $\{AO = vermelho, NGS = vermelho\}$ (que, de acordo com nossa discussão anterior, é inconsistente). Vamos supor que experimentamos *T* = *vermelho* em seguida e depois atribuímos *TN*, *Q*, *V*, *AM*. Sabemos que nenhuma atribuição pode funcionar para essas quatro últimas variáveis e, assim, eventualmente ficamos sem valores para experimentar em *TN*. Agora, a questão é para onde regressar. O retorno não pode funcionar porque *TN tem* valores consistentes com as variáveis atribuídas precedentes — *TN* não tem um conjunto de conflito completo de variáveis precedentes que tenham causado sua falha. Entretanto, sabemos que as quatro variáveis *TN*, *Q*, *V* e *AM*, *tomadas em conjunto*, falharam devido a um conjunto de variáveis precedentes, que devem ser as variáveis que entram em conflito direto com as quatro. Isso nos leva a uma noção mais profunda do conjunto de conflito para uma variável como *TN*: é esse conjunto de variáveis precedentes que fez *TN*, *juntamente com quaisquer variáveis subsequentes*, não ter nenhuma solução consistente. Nesse caso, o conjunto é *AO* e *NGS*, e assim o algoritmo deve regressar até *NGS* e saltar sobre Tasmânia. Um algoritmo de retorno que utiliza conjuntos de conflito definidos desse modo é chamado **retorno orientado por conflito**.

Agora devemos explicar como esses novos conjuntos de conflito são calculados. De fato, o método é muito simples. A falha "terminal" de uma ramificação da busca sempre ocorre porque o domínio de uma variável se torna vazio; essa variável tem um conjunto de conflito-padrão. Em nosso exemplo, *AM* falha, e seu conjunto de conflito é (digamos) $\{AO, TN, Q\}$. Recuamos até *Q*, e *Q* *absorve* o conjunto de conflito de *AM* (com exceção do próprio *Q*, é claro) em seu próprio conjunto de conflito direto, que é $\{TN, NGS\}$; o novo conjunto de conflito é $\{AM, TN, NGS\}$. Isto é, não há solução de *Q* em diante, dada a atribuição anterior a $\{AO, TN, NGS\}$. Portanto, regressamos até *TN*, o mais recente desses elementos. *TN* absorve $\{AO, TN, NGS\} - \{TN\}$ em seu próprio conjunto de conflito direto $\{AO\}$, fornecendo $\{AO, NGS\}$ (como afirmamos no parágrafo anterior). Agora, o algoritmo recua até *NGS*, como seria de esperar. Em resumo, seja X_j a variável corrente e seja $conf(X_j)$ seu conjunto de conflito. Se todo valor possível para *Xj* falhar, recue até a variável mais recente X_i em $conf(X_j)$ e defina:

$$\text{conf}(X_i) \leftarrow \text{conf}(X_i) \cup \text{conf}(X_j) - \{X_i\}.$$

Quando chegamos a uma contradição, o retorno pode nos informar até onde se poderia voltar; assim não se perde tempo alterando variáveis que não vão resolver o problema. Mas também

gostaríamos de desembocar no mesmo problema novamente. Quando a busca chega a uma contradição, sabemos que algum subconjunto do conjunto de conflito é responsável pelo problema. O **aprendizado de restrição** é a ideia de encontrar um conjunto mínimo de variáveis do conjunto de conflito que causa o problema. Esse conjunto de variáveis, juntamente com seus valores correspondentes, é chamado de **nada bom**. Em seguida, registramos o nada bom, quer pela inclusão de uma nova restrição para o PSR, quer mantendo um cache separado de nada bons.

Por exemplo, considere o estado {*AO* = vermelho, *TN* = *verde*, *Q* = *azul*} na última linha da Figura 6.6. A verificação prévia pode informar que esse estado é um nada bom porque não há uma atribuição válida para *AM*. Nesse caso particular, o registro do nada bom não iria ajudar porque, uma vez que esse ramo seja podado da árvore de busca, nunca iremos encontrar essa combinação novamente. Mas suponha que a árvore de busca da Figura 6.6 realmente faizesse parte de uma árvore de busca maior que inicialmente atribuía valores para *V* e *T*. Então, valeria a pena registrar {*AO* = *vermelho*, *TN* = *verde*, *Q* = *azul*} como nada bons porque incorreremos no mesmo problema novamente para cada conjunto possível de atribuições para *V* e *T*.

Efetivamente pode-se utilizar nada bons por verificação à frente ou por retorno. O aprendizado de restrição é uma das técnicas mais importantes utilizadas por solucionadores modernos de PSR para alcançar eficiência em problemas complexos.

6.4 BUSCA LOCAL PARA PSRs

Os algoritmos de busca local (veja a Seção 4.1) se mostram muito eficazes na resolução de vários PSRs. Eles utilizam uma formulação de estados completos: o estado inicial atribui um valor a cada variável e a busca altera o valor de uma variável de cada vez. Por exemplo, no problema de oito rainhas (veja a Figura 4.3), o estado inicial poderia ser uma configuração aleatória de oito rainhas em oito colunas e cada etapa moveria uma única rainha para uma nova posição em sua coluna. Normalmente, o palpite inicial viola várias restrições. O ponto de busca local é eliminar as restrições violadas.[2]

Na escolha de um novo valor para uma variável, a heurística mais óbvia é selecionar o valor que resulta no número mínimo de conflitos com outras variáveis — a heurística de **conflitos mínimos**.

O algoritmo é mostrado na Figura 6.8; sua aplicação a um problema de oito rainhas é diagramada na Figura 6.9.

```
função CONFLITOS-MÍNIMOS(psr, max_etapas) retorna uma solução ou falha
  entradas: psr, um problema de satisfação de restrições
        max_etapas, o número de etapas permitidas antes de desistir

  corrente ← uma atribuição inicial completa para psr
  para i = 1 para max_etapas faça
    se corrente é uma solução para psr então retornar corrente
    var ← uma variável em conflito escolhida ao acaso a partir de VARIÁVEIS[psr]
    valor ← o valor v para var que minimiza CONFLITOS(var, v, corrente, psr)
```

```
    definir var = valor em corrente
  retornar falha
```

Figura 6.8 O algoritmo CONFLITOS-MÍNIMOS para resolução de PSRs por busca local. O estado inicial pode ser escolhido aleatoriamente ou por meio de um processo de atribuição gulosa que escolhe um valor de conflito mínimo para uma variável de cada vez. A função CONFLITOS conta o número de restrições violadas por um valor específico, dado o restante da atribuição corrente.

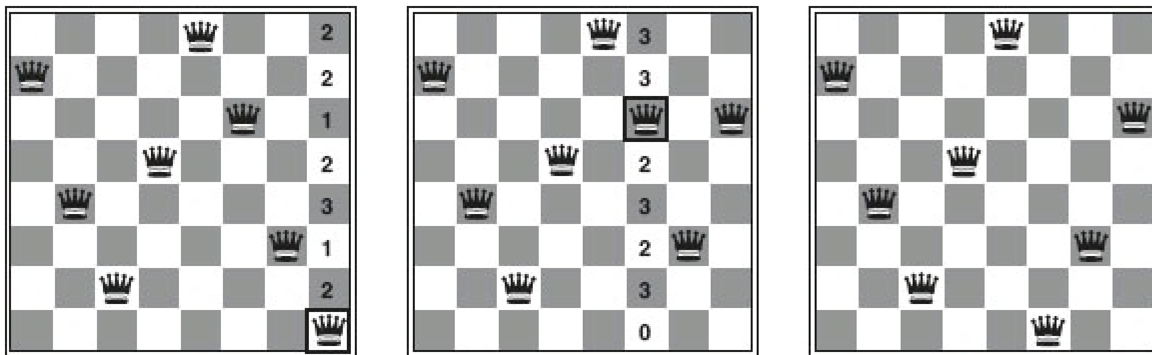


Figura 6.9 Uma solução de duas etapas para um problema de oito rainhas utilizando conflitos mínimos. Em cada fase, é escolhida uma rainha para reatribuição em sua coluna. O número de conflitos (nesse caso, o número de rainhas atacantes) é mostrado em cada quadrado. O algoritmo move a rainha para o quadrado de conflito mínimo, efetuando os desempates ao acaso.

A heurística de conflitos mínimos é surpreendentemente eficaz para muitos PSRs. É incrível observar no problema de *n* rainhas que, se não for levado em conta o posicionamento inicial das rainhas, o tempo de execução de conflitos mínimos será aproximadamente *independente do tamanho do problema*. Essa notável observação foi o estímulo que levou a um intenso estudo da busca local na década de 1990 e à distinção entre problemas fáceis e difíceis, que começamos a examinar no Capítulo 7. Em termos gerais, o problema de *n* rainhas é fácil para a busca local porque as soluções estão densamente distribuídas ao longo do espaço de estados. A heurística de conflitos mínimos também funciona bem para problemas difíceis. Por exemplo, ela é empregada na programação de observações do telescópio espacial Hubble, reduzindo o tempo necessário para programar uma semana de observações de três semanas (!) para algo ao redor de 10 minutos.

Todas as técnicas de busca local a partir da Seção 4.1 são candidatas a ser aplicadas em PSRs, e algumas delas se revelaram especialmente eficazes. O cenário de um PSR sob a heurística de conflitos mínimos geralmente tem uma série de platôs. Pode haver milhões de atribuições de variáveis que estão apenas um conflito distante de uma solução. A busca de platô — permitir movimentos de lado para outro estado com a mesma pontuação — pode ajudar a busca local a encontrar o seu caminho fora desse platô. Essa vagueação no platô pode ser orientada com a **busca de tabu**: manutenção de uma pequena lista de estados visitados recentemente e proibição do retorno do algoritmo para esses estados. Pode-se utilizar também têmpera simulada para escapar do platô.

Outra técnica, chamada **ponderação de restrição**, pode ajudar a concentrar a busca em restrições importantes. É dado a cada restrição um peso numérico, *Wi*; inicialmente todos valem 1. Em cada etapa da busca, o algoritmo escolhe um par de variáveis/valor para alterar o que resultará no menor peso total de todas as restrições violadas. Os pesos são então ajustados, incrementando o peso de

cada restrição que foi violado pela atribuição atual. Isso tem dois benefícios: acrescenta topografia ao platô, certificando-se de que é possível melhorar a partir do estado atual, e também, ao longo do tempo, aumenta o peso das restrições que estão provando serem difíceis de resolver.

Outra vantagem da busca local é a possibilidade de utilizá-la em uma configuração on-line quando o problema se altera. Isso é particularmente importante em problemas de escalonamento. A escala de linhas aéreas para uma semana pode envolver milhares de voos e dezenas de milhares de atribuições de pessoas, mas o mau tempo em um aeroporto pode tornar a escala inviável. Gostaríamos de reparar a escala com um número mínimo de mudanças. Isso pode ser feito com facilidade por meio de um algoritmo de busca local a partir da escala atual. Uma busca com retrocesso com o novo conjunto de restrições normalmente exige muito mais tempo e pode encontrar uma solução com muitas mudanças a partir da escala atual.

6.5 A ESTRUTURA DE PROBLEMAS

Nesta seção, examinaremos meios pelos quais a *estrutura* do problema, representada pelo grafo de restrições, pode ser usada para encontrar soluções com rapidez. A maior parte das abordagens utilizadas aqui também se aplica a outros problemas além de PSRs, tais como o raciocínio probabilístico. Afinal, o único modo de termos esperança de lidar com o mundo real é decompô-lo em muitos subproblemas. Quando observamos mais uma vez a Figura 6.1(b) repetida como Figura 6.12(a)), com a finalidade de identificar a estrutura do problema, um fato se destaca: a Tasmânia não está conectada ao continente.[3] Intuitivamente, é óbvio que colorir a Tasmânia e colorir o continente são **subproblemas independentes** — qualquer solução para o continente combinada a qualquer solução para a Tasmânia produz uma solução para o mapa inteiro. A independência pode ser averiguada simplesmente procurando-se por **componentes conectados** do grafo de restrições. Cada componente corresponde a um subproblema PSR_i. Se a atribuição S_i é uma solução de PSR_i, então $\cup_i S_i$ é uma solução de $\cup_i PSR_i$. Por que isso é importante? Considere o seguinte: vamos supor que cada PSR_i tenha c variáveis do total de n variáveis, onde c é uma constante. Então, existem n/c subproblemas, cada um dos quais exige no máximo o trabalho d^c para ser resolvido, onde d é o tamanho do domínio. Consequentemente, o trabalho total é $O(d^c n/c)$, que é *linear* em n; sem a decomposição, o trabalho total é $O(d^n)$, que é exponencial em n. Vamos tornar esse caso mais concreto: a divisão de um PSR booleano com $n = 80$ variáveis em quatro subproblemas reduz o tempo de solução no pior caso do tempo de duração do universo para menos de um segundo.

Então, subproblemas completamente independentes são interessantes, mas raros. Felizmente, algumas outras estruturas de grafo também são fáceis de resolver. Por exemplo, um grafo de restrição é uma **árvore** quando quaisquer duas variáveis estiverem ligadas por apenas um caminho. Mostraremos que *qualquer PSR estruturado em árvore pode ser resolvido no tempo linear no número de variáveis*.[4] O fundamento é uma nova noção de consistência, chamada **consistência orientada de arco** ou COA. A PSR é definida para ser arco orientado consistente sob uma ordenação de variáveis $X_1, X_2, \ldots, X_n$ se e somente se cada X_i for arco-consistente com cada X_j para $j > i$.

Para resolver um PSR estruturado em árvore, escolha primeiramente qualquer variável para ser a

raiz da árvore e escolha uma ordenação das variáveis de tal forma que cada variável apareça após o seu pai na árvore.

Tal ordenação chama-se **classificação topológica**. A Figura 6.10(a) mostra uma amostra de árvore e (b) mostra uma ordenação possível. Qualquer árvore com n nós tem $n - 1$ arcos, então podemos fazer esse grafo arco-orientado consistente em $O(n)$ etapas, cada uma das quais deve comparar com até d possíveis valores de domínio para duas variáveis, para um tempo total $O(nd^2)$. Uma vez que tenhamos um grafo arco orientado consistente, podemos proceder diretamente à lista de variáveis e escolher algum valor restante. Desde que cada ligação de um pai com seu filho seja arco-consistente, sabemos que, para qualquer valor que escolhamos para o pai, haverá um valor válido deixado a ser escolhido para o filho. Isso significa que não teremos que retroceder, podemos mover-nos linearmente através das variáveis. O algoritmo completo é mostrado na Figura 6.11.

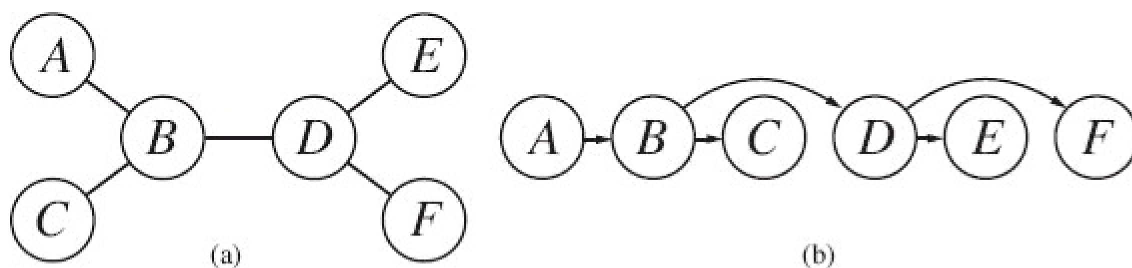


Figura 6.10 (a) O grafo de restrições de um PSR estruturado em árvore. (b) Uma ordenação linear das variáveis consistentes com a árvore, sendo A a raiz. Isso é conhecido como **classificação topológica** de variáveis.

> **função** SOLUCIONADOR-ÁRVORE-PSR(*psr*) **retornar** uma solução ou falha
>   **entradas**: *psr*, um PSR com componentes *X*, *D*, *C*
>
>   *n* ← número de variáveis em X
>   *atribuição* ← uma atribuição vazia
>   *raiz* ← qualquer variável em *X*
>   *X* ← CLASSIFICAÇÃO TOPOLÓGICA (*X*, *raiz*)
>   **para** *j* = *n* **até** 2 **faça**
>   TORNE-ARCO-CONSISTENTE(PAI(X_j),X_j)
>   **se** não puder se tornar consistente **então retornar** *falha*
>   **para** *i* = 1 **até** *n* **faça**
>   *atribuição*[X_i] ← qualquer valor consistente de D_i
>   **se** não houver valor consistente **então retornar** *falha*
>   **retornar** *atribuição*

Figura 6.11 O algoritmo SOLUCIONADOR-ÁRVORE-PSR para resolver PSRs estruturados em árvore. Se o PSR tiver uma solução, vamos encontrá-la em tempo linear; senão, detectaremos uma contradição.

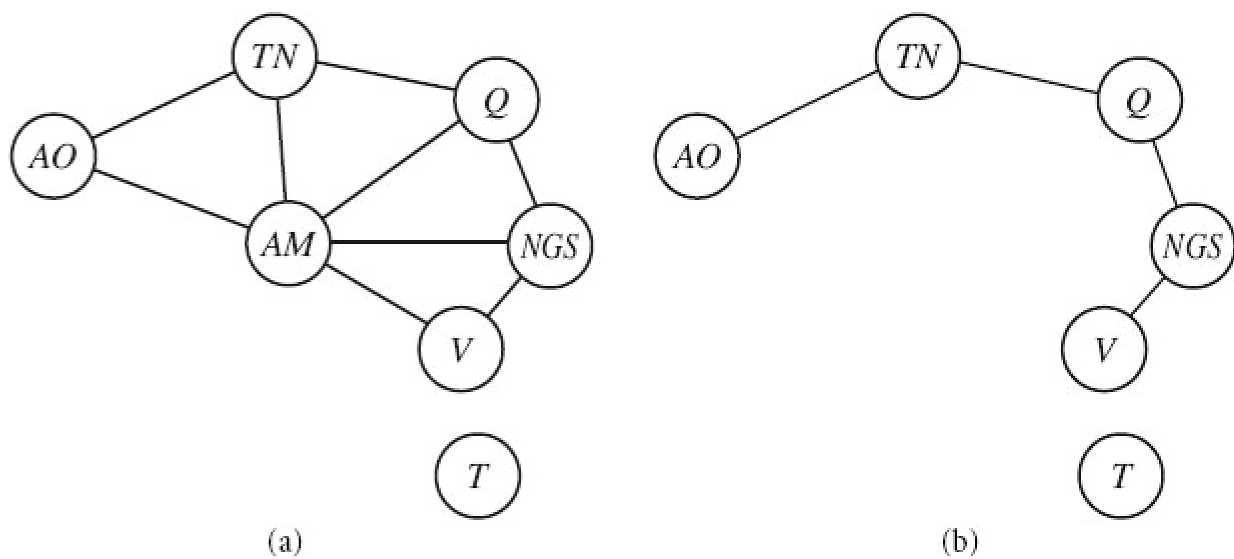


Figura 6.12 (a) Grafo de restrições original da Figura 6.1. (b) Grafo de restrições após a remoção de *AM*.

Agora que temos um algoritmo eficiente para árvores, podemos considerar se grafos de restrições mais gerais podem ser *reduzidos* de algum modo a árvores. Há duas alternativas principais para fazer isso, uma delas baseada na remoção de nós e outra baseada na condensação de nós.

A primeira abordagem envolve a atribuição de valores a algumas variáveis, de modo que as variáveis restantes formem uma árvore. Considere o grafo de restrições para a Austrália, mostrado novamente na Figura 6.12(a). Se pudéssemos eliminar Austrália meridional, o grafo se tornaria uma árvore, como em (b). Felizmente, podemos fazer isso (no grafo, não no continente) fixando um valor para *AM* e excluindo dos domínios das outras variáveis quaisquer valores inconsistentes com o valor escolhido para *AM*.

Agora, qualquer solução para o PSR depois que *AM* e suas restrições forem removidas será consistente com o valor escolhido para *AM* (isso funciona para PSRs binários; a situação é mais complicada com restrições de alta ordem). Então, podemos resolver a árvore restante com o algoritmo dado anteriormente e, desse modo, resolver o problema inteiro. É claro que, no caso geral (em vez de coloração de mapas), o valor escolhido para *AM* poderia ser o valor errado e, assim, precisaríamos experimentar cada um deles. O algoritmo geral é:

1. Escolha um subconjunto *S* de variáveis PSR tal que o grafo de restrições se torne uma árvore depois da remoção de *S*. *S* é chamado de **conjunto de corte de ciclo**.
2. Para cada atribuição possível às variáveis de *S* que satisfaça a todas as restrições sobre *S*,
   (a) remova dos domínios das variáveis restantes quaisquer valores que sejam inconsistentes com a atribuição para *S* e
   (b) se o PSR restante tiver uma solução, retorne-a juntamente com a atribuição para *S*.

Se o conjunto de corte de ciclo tiver tamanho *c*, o tempo de execução total será $O(d^c \cdot (n - c) d^2)$: temos que experimentar cada uma das combinações d^c de valores para as variáveis em *S* e, para cada combinação, devemos resolver um problema de árvore de tamanho $n - c$. Se o grafo for "praticamente uma árvore", então *c* será pequeno e as economias em relação ao retrocesso direto serão enormes. Porém, no pior caso, *c* poderá chegar a $(n - 2)$. Encontrar o *menor* conjunto de corte

de ciclo será NP-difícil, mas são conhecidos diversos algoritmos de aproximação eficientes para essa tarefa. A abordagem algorítmica global é chamada **condicionamento de conjunto de corte**; surgirá novamente no Capítulo 14, onde ela é utilizada no raciocínio probabilístico.

A segunda abordagem se baseia na construção de uma **decomposição em árvore** do grafo de restrições em um conjunto de subproblemas conectados. Cada subproblema é resolvido independentemente, e as soluções resultantes são então combinadas. Como a maioria dos algoritmos de dividir e conquistar, isso funciona bem se nenhum subproblema é muito grande. A Figura 6.13 mostra uma decomposição em árvore do problema de coloração de mapa em cinco subproblemas. Uma decomposição em árvore deve satisfazer os três requisitos a seguir:

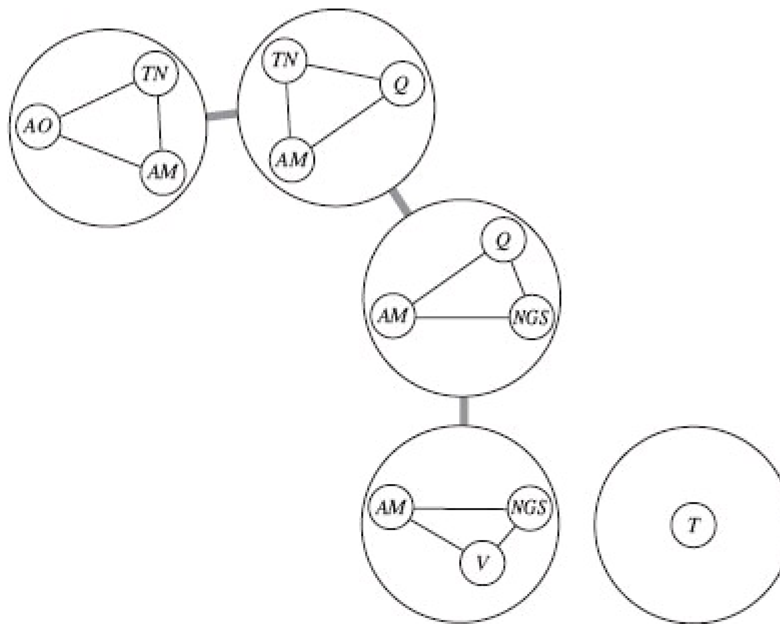


Figura 6.13 Decomposição em árvore do grafo de restrições da Figura 6.12(a).

- Toda variável no problema original aparece em pelo menos um dos subproblemas.
- Se duas variáveis estiverem conectadas por uma restrição no problema original, elas deverão aparecer juntas (e juntamente com a restrição) em pelo menos um dos subproblemas.
- Se uma variável aparecer em dois subproblemas na árvore, ela deverá aparecer em todo subproblema ao longo do caminho que conecta esses dois subproblemas.

As duas primeiras condições garantem que todas as variáveis e restrições estarão representadas na decomposição. A terceira condição parece bastante técnica, mas simplesmente reflete a restrição de que qualquer variável dada deve ter o mesmo valor em todo subproblema em que aparece; os vínculos que unem subproblemas na árvore impõem essa restrição. Por exemplo, *AM* aparece em todos os quatro subproblemas conectados da Figura 6.13. Você poderá verificar pela Figura 6.12 que essa decomposição faz sentido.

Resolvemos cada subproblema independentemente; se qualquer um não tiver solução, saberemos que o problema inteiro não tem solução. Se conseguirmos resolver todos os subproblemas, tentaremos construir uma solução global como a seguir. Primeiro, visualizamos cada subproblema como uma “megavariável” cujo domínio é o conjunto de todas as soluções para o subproblema. Por exemplo, o subproblema mais à esquerda na Figura 6.13 é um problema de coloração de mapa com três variáveis e, consequentemente, tem seis soluções — uma delas é {*AO* = *vermelho*, *AM* = *azul*,

TN = *verde*}. Em seguida, resolvemos as restrições que conectam os subproblemas com a utilização do algoritmo eficiente para árvores apresentado antes. As restrições entre subproblemas simplesmente insistem que as soluções de subproblemas concordem sobre suas variáveis compartilhadas. Por exemplo, dada a solução {*AO* = *vermelho*, *AM* = *azul*, *TN* = *verde*} para o primeiro subproblema, a única solução consistente para o próximo subproblema é {*AM* = *azul*, *TN* = *verde*, *Q* = *vermelho*}.

Um dado grafo de restrições admite muitas decomposições em árvore; ao se escolher uma decomposição, o objetivo é tornar os subproblemas tão pequenos quanto possível. A **largura de árvore** de uma decomposição em árvore de um grafo é uma unidade menor que o tamanho do maior subproblema; a largura de árvore do grafo propriamente dito é definida como a mínima largura de árvore entre todas as suas decomposições em árvore. Se um grafo tem largura de árvore w e recebemos a decomposição em árvore correspondente, então o problema pode ser resolvido no tempo $O(nd^{w+1})$. Consequentemente, *PSRs que têm grafos de restrições com largura de árvore limitada podem ser resolvidos em tempo polinomial*. Infelizmente, encontrar a decomposição em árvore com largura de árvore mínima é um problema NP-difícil, mas existem métodos heurísticos que funcionam bem na prática.

Até agora, vimos a estrutura de grafo de restrição. Pode haver também uma estrutura importante nos *valores* das variáveis. Considere o problema de coloração do mapa com n cores. Para cada solução consistente, há realmente um conjunto de $n!$ soluções formadas pela permuta dos nomes das cores. Por exemplo, no mapa da Austrália sabemos que *AO*, *TN* e *AS* devem ter cores diferentes, mas existem 3! = 6 maneiras de atribuir as três cores para essas três regiões. Isso é chamado de **simetria de valor**. Gostaríamos de reduzir o espaço de busca a um fator de $n!$ quebrando a simetria. Fazemos isso através da introdução de uma **restrição de quebra de simetria**. Para o nosso exemplo, poderíamos impor uma restrição arbitrária de ordenação, $TN < AM < AO$, que exige que os três valores estejam em ordem alfabética. Essa restrição garante que apenas uma das $n!$ soluções seja possível: {*TN* = *azul, AM* = *verde, AO* = *vermelho*}.

Para a coloração do mapa, foi fácil encontrar uma restrição que eliminasse a simetria e, em geral, é possível encontrar restrições que eliminam todas, em vez de uma solução simétrica em tempo polinomial, mas é NP-difícil eliminar todas as simetrias entre os conjuntos intermediários de valores durante a pesquisa. Na prática, quebrar o valor da simetria provou ser importante e eficaz em uma ampla gama de problemas.

6.6 RESUMO

- Os **problemas de satisfação de restrições** (ou PSRs) representam um estado com um conjunto de variáveis/valores pares e representam as condições para uma solução por um conjunto de restrições sobre as variáveis. Muitos problemas importantes do mundo real podem ser descritos como PSRs.
- Várias técnicas de inferência usam as restrições para inferir quais variáveis/pares de valores são consistentes e quais não são. Elas incluem nó, arco, caminho e *k*-consistência.
- A **busca com retrocesso**, uma forma de busca em profundidade, é comumente usada para

resolver PSRs. A inferência pode ser interligada com a busca.

- As heurísticas de **valores restantes mínimos** e de **grau** são métodos independentes do domínio para decidir que variável escolher em seguida em uma busca com retrocesso. A heurística de **valor menos restritivo** ajuda a decidir que valor se deve experimentar primeiro para determinada variável. O retrocesso ocorre quando não é possível encontrar nenhuma atribuição válida para uma variável. O **retorno orientado por conflito** efetua o retrocesso diretamente para a origem do problema.
- A busca local utilizando a heurística de **conflitos mínimos** tem sido aplicada com grande sucesso a problemas de satisfação de restrições.
- A complexidade da resolução de um PSR está fortemente relacionada à estrutura de seu grafo de restrições. Problemas estruturados em árvore podem ser resolvidos em tempo linear. O **condicionamento de conjunto de corte** pode reduzir um PSR geral a um PSR estruturado em árvore e é muito eficiente no caso de ser possível encontrar um conjunto de corte pequeno. As técnicas de **decomposição em árvore** transformam o PSR em uma árvore de subproblemas e são eficientes quando a **largura de árvore** do grafo de restrições é pequena.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O trabalho mais antigo relacionado à satisfação de restrições lidava em grande parte com restrições numéricas. As restrições de equações com domínios de inteiros foram estudadas pelo matemático indiano Brahmagupta no século VII; com frequência, elas se denominam **equações diofantinas**, em homenagem ao matemático grego Diofanto (c. 200-284), que realmente considerou o domínio de racionais positivos. Métodos sistemáticos para resolução de equações lineares por eliminação de variáveis foram estudados por Gauss (1829); a solução de restrições de desigualdade lineares teve início com Fourier (1827).

Os problemas de satisfação de restrições de domínios finitos também têm uma longa história. Por exemplo, a **coloração de grafo** (da qual a coloração de mapa é um caso especial) é um problema antigo em matemática. A conjetura das quatro cores (de que todo grafo planar pode ser colorido com quatro cores ou menos) foi elaborada primeiro por Francis Guthrie, aluno do matemático De Morgan, em 1852. Ela resistiu à solução — apesar de diversas afirmações publicadas em contrário — até ser criada uma prova, por Appel e Haken (1977) [consulte o livro *Four Colors Suffice* (Wilson, 2004)]. Os puristas ficaram desapontados porque parte da prova fiou-se em um computador, assim Georges Gonthier (2008), usando o demonstrador de teorema de COQ, derivou uma prova formal de que a prova de Appel e Haken estava correta.

Classes específicas de problemas de satisfação de restrições ocorrem em toda a história da ciência da computação. Um dos exemplos mais influentes foi o sistema SKETCHPAD (Sutherland, 1963), que resolvia restrições geométricas em diagramas e foi o precursor dos modernos programas de desenho e das ferramentas de CAD. A identificação de PSRs como classe *geral* se deve a Ugo Montanari (1974). A redução de PSRs de alta ordem a PSRs puramente binários com variáveis auxiliares (veja o Exercício 6.6) foi realizada originalmente pelo lógico do século XIX Charles Sanders Peirce. Ela foi introduzida na literatura de PSRs por Dechter (1990b) e elaborada por

Bacchus e Van Beek (1998). PSRs com preferências entre soluções são estudados amplamente na literatura de otimização; veja em Bistarelli *et al*. (1997) uma generalização da estrutura de PSRs para permitir o uso de preferências. O algoritmo de eliminação de compartimentos (Dechter, 1999) também pode ser aplicado a problemas de otimização.

Métodos de propagação de restrições foram popularizados pelo sucesso de Waltz (1975) em problemas de linha de rotulagem poliédrica para visão de computador. Waltz mostrou que, em muitos problemas, a propagação elimina completamente a necessidade de retrocesso. Montanari (1974) introduziu a noção de redes de restrição e propagação pela consistência de caminho. Alan Mackworth (1977) propôs o algoritmo AC-3 para reforçar a consistência de arco, bem como a ideia geral de combinação de retrocesso com algum grau de aplicação da consistência. Mohr e Henderson (1986) desenvolveram o AC-4, um algoritmo mais consistente de consistência de arco. Logo depois apareceu o trabalho de Mackworth, e os pesquisadores começaram a fazer experiência com a contrapartida entre o custo da aplicação da consistência e os benefícios em termos de redução de busca. Haralick e Elliot (1980) favoreceram o algoritmo mínimo de verificação prévia descrito por McGregor (1979), enquanto Gaschnig (1979) sugeriu total consistência de arco após cada atribuição de variável — um algoritmo chamado mais tarde de MAC por Sabin e Freuder (1994). O último documento forneceu evidências convincentes de que em PSRs mais difíceis, cheios de consistência de arcos, a verificação compensa. Freuder (1978, 1982) investigou a noção de *k*-consistência e sua relação com a complexidade em resolver PSRs. Apt (1999) descreveu uma estrutura de algoritmo genérica em que os algoritmos de propagação de consistência podem ser analisados e, antes, Bessière (2006) apresentou uma avaliação atualizada.

Os métodos especiais para manipulação de restrições de alta ordem ou globais foram desenvolvidos primeiramente dentro do contexto de **programação de lógica de restrições**. Marriott e Stuckey (1998) fornecem excelente cobertura de buscas nessa área. A restrição *TodosDiferentes* foi estudada por Regin (1994), Stergiou e Walsh (1999) e Van Hoeve (2001). As restrições de limites foram incorporadas à programação de lógica de restrições por Van Hentenryck *et al*. (1998). Uma pesquisa de restrições globais foi fornecida por Van Hoeve e Katriel (2006).

O Sudoku tornou o PSR mais conhecido e assim foi descrito por Simonis (2005). Agerbeck e Hansen (2008) descrevem algumas das estratégias e mostram que o Sudoku em um tabuleiro $n^2 \times n^2$ está na classe dos problemas NP-difíceis. Reeson et al. (2007) fazem uma demonstração iterativa baseada em técnicas PSR.

A ideia de busca por retrocesso remonta a Golomb e Baumert (1965), e sua aplicação para a satisfação de restrição deve-se a Bitner e Reingold (1975), embora trace o algoritmo básico do século XIX. Bitner e Reingold também introduziram a heurística VRM, que eles chamavam de heurística de variável mais restrita. Brelaz (1979) usou a heurística de grau como desempate após a aplicação da heurística VRM. O algoritmo resultante, apesar da sua simplicidade, ainda é o melhor método para grafos arbitrários de *k*-coloração. Haralick e Elliot (1980) propuseram a heurística do valor menos restritivo.

O método básico de retorno (*backjumping*) é devido a John Gaschnig (1977, 1979). Kondrak e Van Beek (1997) mostraram que esse algoritmo foi essencialmente incorporado pela verificação prévia. O retorno orientado por conflito foi criado por Prosser (1993). Na realidade, a forma mais geral e poderosa de retrocesso inteligente foi desenvolvida muito cedo por Stallman e Sussman

(1977). Sua técnica de **retrocesso orientado por dependência** levou ao desenvolvimento de **sistemas de manutenção de verdade** (Doyle, 1979), que discutiremos na Seção 12.6.2. A conexão entre as duas áreas é analisada por De Kleer (1989).

O trabalho de Stallman e Sussman também introduziu a ideia de **aprendizagem de restrição**, em que resultados parciais obtidos por busca podem ser gravados e reutilizados mais tarde na busca. A ideia foi apresentada formalmente na busca com retrocesso formalizada por Dechter (1990a). A **marcação para trás** (Gaschnig, 1979) é um método particularmente simples no qual as atribuições consistentes e inconsistentes são gravadas aos pares e usadas para evitar a repetição de verificações de restrições. A marcação para trás pode ser combinada com o retorno orientado por conflito; Kondrak e Van Beek (1997) apresentam um algoritmo híbrido que provavelmente inclui qualquer método tomado separadamente. O método de **retrocesso dinâmico** (Ginsberg, 1993) retém atribuições parciais bem-sucedidas de subconjuntos de variáveis posteriores ao realizar o retrocesso sobre uma escolha anterior que não invalida o sucesso posterior.

Estudos empíricos de vários métodos de retrocesso randomizado foram feitos por Gomes *et al.* (2000) e Gomes e Selman (2001). Van Beek (2006) pesquisou o retrocesso.

A busca local em problemas de satisfação de restrições foi popularizada pelo trabalho de Kirkpatrick *et al.* (1983) sobre têmpera simulada (veja o Capítulo 4), amplamente utilizado em problemas de escalonamento. A heurística de conflitos mínimos foi proposta inicialmente por Gu (1989) e de forma independente por Minton *et al.* (1992). Sosic e Gu (1994) mostraram como ela poderia ser aplicada para resolver o problema de 3.000.000 de rainhas em menos de um minuto. O espantoso sucesso da busca local utilizando conflitos mínimos de *n* rainhas levou a uma reavaliação da natureza e da prevalência de problemas “fáceis” e “difíceis”. Peter Cheeseman *et al.* (1991) exploraram a dificuldade de PSRs gerados aleatoriamente e descobriram que quase todos esses problemas são trivialmente fáceis ou não têm nenhuma solução. Apenas se os parâmetros do gerador de problemas forem definidos em certo intervalo estreito, dentro do qual aproximadamente metade dos problemas é solúvel, encontraremos instâncias de problemas “difíceis”. Discutiremos esse fenômeno com mais detalhes no Capítulo 7. Konolige (1994) mostrou que a busca local é inferior à busca em retrocesso para problemas com certo grau de estrutura local, o que levou a um trabalho que combinou busca local e inferência, como o de Pinkas e Dechter (1995). Hoos e Tsang (2006) pesquisaram técnicas de busca local.

O trabalho relacionado à estrutura e à complexidade de PSRs teve origem em Freuder (1985), que mostrou que a busca em árvores com consistência de arco funciona sem qualquer retrocesso. Um resultado semelhante, com extensões para hipergrafos acíclicos, foi desenvolvido na comunidade de bancos de dados (Beeri *et al.*, 1983). Bayardo e Miranker (1994) apresentaram um algoritmo para PSRs estruturado em árvore que executa em tempo linear sem qualquer pré-processamento.

Desde que esses artigos foram publicados, houve um grande progresso no desenvolvimento de resultados mais gerais relacionando a complexidade da resolução de um PSR à estrutura de seu grafo de restrições. A noção de largura de árvore foi introduzida pelos teóricos de grafo Robertson e Seymour (1986). Dechter e Pearl (1987, 1989), fundamentados no trabalho de Freuder, aplicaram a mesma noção (que denominaram **largura induzida**) a problemas de satisfação de restrições e desenvolveram a abordagem de decomposição em árvore descrita na Seção 6.5. Com base nesse trabalho e nos resultados da teoria de bancos de dados, Gottlob *et al.* (1999a, 1999b) desenvolveram

a noção de **largura de hiperárvore**, baseada na caracterização do PSR como um hipergrafo. Além disso, para mostrar que qualquer PSR com largura de hiperárvore w pode ser resolvido no tempo $O(n^{w+1} \log n)$, eles também demonstraram que a largura de hiperárvore inclui todas as medidas definidas anteriormente de "largura", no sentido de que existem casos em que a largura de hiperárvore é limitada, e as outras medidas são ilimitadas.

O interesse em rememorar as abordagens anteriores de retrocesso foi reacendido pelo trabalho de Bayardo e Schrag (1997), cujo algoritmo RELSAT combinou aprendizagem de restrição e retorno, e mostrou que superava muitos outros algoritmos da época. Isso levou aos algoritmos de busca E/OU aplicáveis tanto a PSRs quanto ao raciocínio probabilístico (Dechter e Mateescu, 2007). Brown *et al*. (1988) introduziram a ideia de quebra de simetria em PSR, e Gent *et al.* (2006) ofereceram uma pesquisa recente.

O campo de **satisfação de restrição distribuída** observa a resolução de PSR quando há uma coleção de agentes, cada qual controlando um subconjunto das variáveis de restrição. Desde 2000 realizam-se *workshops* anuais sobre esse problema e existe ampla abordagem por toda parte (Collin *et al*., 1999; Pearce *et al*., 2008; Shoham e Leyton-Brown, 2009).

A comparação de algoritmos PSR é principalmente uma ciência empírica: alguns resultados teóricos mostram que um algoritmo domina um outro sobre todos os problemas; em lugar disso, precisamos executar tentativas para ver quais algoritmos têm melhor desempenho em situações de instância de problemas típicos. Como Hooker (1995) apontou, precisamos ter cuidado em distinguir entre testes competitivos, como ocorre em competições entre os algoritmos baseados em tempo de execução, e testes científicos, cujo objetivo é identificar as propriedades de um algoritmo que determinam a sua eficácia em uma classe de problemas.

Os livros didáticos recentes por Apt (2003) e Dechter (2003), e a coleção de Rossi *et al.* (2006) são excelentes recursos de processamento de restrição. Existem diversos trabalhos de pesquisa de boa qualidade sobre técnicas de PSRs, incluindo os de Kumar (1992), Dechter e Frost (1999), e Bartak (2001), além dos excelentes artigos de Dechter (1992) e Mackworth (1992). Pearson e Jeavons (1997) pesquisam classes tratáveis de PSRs, cobrindo tanto os métodos de decomposição estrutural quanto métodos que se baseiam em propriedades dos domínios ou das próprias restrições.

Kondrak e Van Beek (1997) fornecem um estudo analítico dos algoritmos de busca com retrocesso, e Bacchus e Van Run (1995) apresentam um estudo mais empírico. A programação de restrição é abordada nos livros de Apt (2003) e Fruhwirth e Abdennadher (2003). Artigos sobre satisfação de restrições aparecem regularmente em *Artificial Intelligence* e no periódico especializado *Constraints*. A principal conferência é a International Conference on Principles and Practice of Constraint Programming, frequentemente chamada *CP*.

EXERCÍCIOS

6.1 Quantas soluções existem para o problema de coloração de mapa da Figura 6.1? Quantas soluções existem se quatro cores forem permitidas? Duas cores?

6.2 Considere o problema de colocar k cavalos em um tabuleiro de xadrez n x n tal que dois cavalos

não se ataquem mutuamente, onde k é dado e $k \leq n^2$.

a. Escolha uma formulação PSR. Nessa formulação, quais são as variáveis?

b. Quais são os valores possíveis de cada variável?

c. Qual o conjunto de variáveis que são restritas e como?

d. Agora considere o problema de colocar *tantos cavalos quanto possível* no tabuleiro sem que eles ataquem. Explique como resolver os problemas através de busca local definindo funções AÇÕES e RESULTADO apropriadas e uma função objetivo sensível.

6.3 Considere o problema de construir (não de resolver) quebra-cabeças de palavras cruzadas:[5] encaixar palavras em uma grade retangular. A grade, dada como parte do problema, especifica quais quadrados estão vazios e quais deles estão sombreados. Suponha que uma lista de palavras (isto é, um dicionário) seja fornecida e que a tarefa seja preencher os quadrados vazios usando qualquer subconjunto da lista. Formule esse problema de forma exata, de duas maneiras:

a. Como um problema de busca geral. Escolha um algoritmo de busca apropriado e especifique uma função heurística, se achar que é necessário utilizá-la. É melhor preencher os espaços vazios uma letra de cada vez ou uma palavra de cada vez?

b. Como um problema de satisfação de restrições. As variáveis devem ser palavras ou letras?

Na sua opinião, qual será a melhor formulação? Por quê?

6.4 Forneça formulações precisas para cada um dos problemas a seguir como problemas de satisfação de restrições:

a. Criação de planta-baixa retilínea: encontrar posições não superpostas em um retângulo grande para vários retângulos menores.

b. Escalonamento de aulas: existe um número fixo de professores e salas de aula, uma lista de aulas a serem oferecidas e uma lista de tempos vagos possíveis para as aulas. Cada professor tem um conjunto de aulas que pode ministrar.

c. Ciclo hamiltoniano: dada uma rede de cidades ligadas por estradas, escolha uma ordem para visitar todas as cidades em um país sem repetir nenhuma.

6.5 Resolva o problema criptoaritmético da Figura 6.2 à mão, utilizando a estratégia de retrocesso com verificação prévia e de VRM e valor menos restritivo.

6.6 Mostre que uma única restrição ternária do tipo “$A + B = C$” pode ser transformada em três restrições binárias usando-se uma variável auxiliar. Suponha domínios finitos. (*Sugestão:* Considere uma nova variável que assume valores que são pares de outros valores e considere restrições como “*X* é o primeiro elemento do par *Y*”.) Em seguida, mostre que restrições com mais de três variáveis podem ser tratadas de modo semelhante. Finalmente, mostre que restrições unárias podem ser eliminadas alterando-se os domínios de variáveis. Isso completa a demonstração de que qualquer PSR pode ser transformado em um PSR apenas com restrições binárias.

6.7 Considere o seguinte quebra-cabeça lógico: em cinco casas, cada uma com uma cor diferente, moram cinco pessoas de diferentes nacionalidades, cada uma das quais prefere uma marca de cigarros diferente, uma bebida diferente e um animal de estimação diferente. Dados os fatos a seguir,

a pergunta a responder é: “Onde vive a zebra e em que casa se bebe água?”

O inglês mora na casa vermelha.
O espanhol é o dono do cachorro.
O norueguês mora na primeira casa à esquerda.
A casa verde é imediatamente à direita da casa marfim.
O homem que come barras Hersey vive na casa ao lado do homem com a raposa.
Kit Kats são consumidos na casa amarela.
O norueguês mora ao lado da casa azul.
O consumidor de Smarties come a própria unha.
O consumidor de Snickers bebe suco de laranja.
O ucraniano bebe chá.
O japonês come Milky Ways.
Kit Kats são consumidos na casa ao lado de onde o cavalo é mantido.
Bebe-se café na casa verde.
Bebe-se leite na casa do meio.

Descreva diferentes representações desse problema como um PSR. Por que seria preferível uma representação em vez de outra?

6.8 Considere o grafo com oito nós de A_1, A_2, A_3, A_4, H, T, F_1, F_2. A_i está ligado em A_{i+1} para todo i, cada A_i está ligado a H, H está ligado a T e T está ligado a cada F_i. Encontre três colorações desse grafo à mão usando a seguinte estratégia: retrocesso com retorno orientado por conflito, a ordem das variáveis A_1, H, A_4, F_1, A_2, F_2, A_3, T e a ordem dos valores R, G, B.

6.9 Explique por que é uma boa heurística escolher a variável *mais* restrita, mas selecionar o valor *menos* restritivo em uma busca de PSR.

6.10 Gere instâncias aleatórias dos problemas de coloração do mapa da seguinte forma: espalhe n pontos em uma unidade quadrada; selecione um ponto X de forma aleatória, conecte X por uma linha reta ao ponto mais próximo de Y tal que X não esteja conectado a Y e a linha não atravesse nenhuma outra linha; repita a etapa anterior até que não seja possível mais conexões. Os pontos representam as regiões no mapa e as linhas ligam-se aos vizinhos. Agora tente encontrar k-colorações de cada mapa, para $k = 3$ e $k = 4$, utilizando conflitos mínimos, retrocesso, retrocesso com verificação prévia e retrocesso com MAC. Construa uma tabela de tempos médios de execução para cada algoritmo para valores de n até o maior que você possa gerenciar. Comente sobre seus resultados.

6.11 Use o algoritmo CA-3 para mostrar que a consistência de arco é capaz de detectar a inconsistência da atribuição parcial $\{AO = verde, V = \text{vermelho}\}$ para o problema mostrado na Figura 6.1.

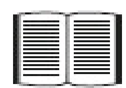
6.12 Qual é a complexidade do pior caso da execução de CA-3 sobre um PSR estruturado em árvore?

6.13 CA-3 coloca de volta na fila *todo* arco (X_k, X_i) sempre que *qualquer* valor é excluído do domínio de X_i, mesmo que cada valor de X_k seja consistente com diversos valores restantes de X_i. Vamos supor que, para todo arco (X_k, X_i), mantemos o controle do número de valores restantes de X_i que são consistentes com cada valor de X_k. Explique como atualizar esses números de modo eficiente e, consequentemente, mostre que a consistência de arco pode ser imposta no tempo total $O(n^2d^2)$.

6.14 O SOLUCIONADOR-ÁRVORE-PSR (Figura 6.10) faz com que a consistência de arco inicie a partir de folhas e volte em direção à raiz. Por que ele faz isso? O que aconteceria se ele fosse para a direção oposta?

6.15 Introduzimos o Sudoku como um PSR a ser resolvido por busca sobre atribuições parciais porque essa é a maneira como as pessoas geralmente se comprometem com a resolução de problemas de Sudoku. Também é possível, certamente, atacar esses problemas com busca local complementando essas atribuições. Como se sairia um solucionador local usando a heurística de conflitos mínimos em problemas do Sudoku?

6.16 Defina com suas próprias palavras as expressões problema de satisfação de restrições, restrição, busca com retrocesso, consistência de arco, retorno e conflitos mínimos.

 **6.17** Suponha que se saiba que um grafo tem um conjunto de corte de ciclo de não mais de k nós. Descreva um algoritmo simples para encontrar um conjunto de corte de ciclo mínimo cujo tempo de execução não seja muito maior que $O(n^k)$ para um PSR com n variáveis. Pesquise na literatura métodos para encontrar conjuntos de corte de ciclos aproximadamente mínimos em um tempo polinomial no tamanho do conjunto de corte. A existência de tais algoritmos torna prático o método de conjunto de corte de ciclo?

---

[1] Um algoritmo AC-4 (Mohr e Henderson, 1986) executa no pior caso de tempo $O(cd^2)$, mas pode ser mais lento do que AC-3 na maioria dos casos. Veja o Exercício 6.13.

[2] A busca local pode ser estendida facilmente para problemas de otimização de restrições PORs. Nesse caso, todas as técnicas de subida de encosta e têmpera simulada podem ser aplicadas para otimizar a função objetivo.

[3] Um cartógrafo cuidadoso ou um tasmaniano patriota poderia objetar que a Tasmânia não deve ter a mesma cor de seu vizinho.

[4] Infelizmente, bem poucas regiões do mundo têm mapas estruturados em árvore, apesar de Sulawesi chegar perto.

[5] Ginsberg *et al*. (1990) descrevem diversos métodos para construir quebra-cabeças de palavras cruzadas. Littman *et al*. (1999) atacam o problema mais difícil de resolvê-los.

PARTE III

Conhecimento, pensamento e planejamento

CAPÍTULO 7

Agentes lógicos

Em que projetamos agentes que podem formar representações de um mundo complexo, usar um processo de inferência para derivar novas representações sobre o mundo e utilizar essas novas representações para deduzir o que fazer.

Os seres humanos, parece, sabem das coisas; e o que sabem os ajuda a fazer coisas. Essas não são declarações vazias. Produzem alegações fortes de como a inteligência dos seres humanos é alcançada — não por mecanismos puramente reflexos, mas por processos de **raciocínio** que operam em **representações** internas do conhecimento. Em IA, essa abordagem à inteligência é incorporada em **agentes baseados em conhecimento**.

Os agentes de resolução de problemas dos Capítulos 3 e 4 conhecem as coisas, mas apenas em um sentido muito limitado, inflexível. Por exemplo, o modelo de transição do quebra-cabeça de oito peças — conhecimento do que as ações fazem — está escondido dentro do código de domínio específico da função RESULTADO. Pode ser utilizado para prever o resultado das ações, mas não para deduzir que duas peças não podem ocupar o mesmo espaço ou que estados com paridade ímpar não podem ser alcançados de estados com paridade par. As representações atômicas utilizadas por agentes de resolução de problemas também são muito limitantes. Em um ambiente parcialmente observável, a única escolha do agente para representar o que sabe sobre o estado atual é listar todos os possíveis estados concretos — uma perspectiva sem esperança em grandes ambientes.

O Capítulo 6 introduziu a ideia de representação dos estados como atribuições de valores para as variáveis; esse é um passo na direção certa, permitindo que algumas partes do agente trabalhem em uma forma de domínio independente e permitindo algoritmos mais eficientes. Neste capítulo e naqueles que se seguem, tomaremos essa etapa na sua conclusão lógica, por assim dizer — desenvolveremos a **lógica** como uma classe geral de representações para apoiar agentes baseados no conhecimento. Tais agentes poderão combinar e recombinar informações para atender às finalidades inumeráveis. Muitas vezes, esse processo pode ser bastante distante das necessidades do momento — como quando um matemático demonstra um teorema ou um astrônomo calcula a expectativa de vida da Terra. Os agentes baseados no conhecimento são capazes de aceitar novas tarefas sob a forma de metas descritas de modo explícito, podem alcançar competência rapidamente ao serem informados ou ao adquirirem novos conhecimentos sobre o ambiente e podem se adaptar a mudanças no ambiente, atualizando o conhecimento relevante.

Começamos na Seção 7.1 com o projeto global de agentes. A Seção 7.2 introduz um novo ambiente

simples, o mundo de wumpus, e ilustra a operação de um agente baseado em conhecimento sem entrar em qualquer detalhe técnico. Em seguida, explicamos os princípios gerais da **lógica** na Seção 7.3 e o caso particular da **lógica proposicional** na Seção 7.4. Embora seja muito menos expressiva que a **lógica de primeira ordem** (Capítulo 8), a lógica proposicional serve para ilustrar todos os conceitos básicos de lógica; é também acompanhada de tecnologias de inferência bem desenvolvidas, que descreveremos nas Seções 7.5 e 7.6. Finalmente, a Seção 7.7 combina o conceito de agentes baseados em conhecimento com a tecnologia de lógica proposicional com a finalidade de construir alguns agentes simples para o mundo de wumpus.

7.1 AGENTES BASEADOS EM CONHECIMENTO

O componente central de um agente baseado em conhecimento é sua **base de conhecimento**, ou KB. Informalmente, uma base de conhecimento é um conjunto de **sentenças** (aqui, "sentença" é utilizada como um termo técnico; está relacionada mas não é idêntica às sentenças em português e em outros idiomas ou linguagens naturais). Cada sentença é expressa em uma linguagem chamada **linguagem de representação de conhecimento** e representa alguma asserção sobre o mundo. Às vezes, ilustramos uma sentença com o nome **axioma**, quando a sentença for tomada como dada sem ser derivada de outras sentenças.

Deve haver um modo para adicionar novas sentenças à base de conhecimento e um meio de consultar o que se conhece. Os nomes-padrão para essas operações são TELL (informe) e ASK (pergunte), respectivamente. Ambas as operações podem envolver **inferência**, ou seja, a derivação de novas sentenças a partir de sentenças antigas.

A inferência deve obedecer ao requisito fundamental de que, quando se formula (com ASK) uma pergunta para a base de conhecimento, a resposta deve seguir do que foi informado (com TELL) anteriormente à base de conhecimento. Mais adiante neste capítulo, seremos mais precisos quanto ao termo fundamental "seguir". No momento, basta levar em conta que ele significa que o processo de inferência não deve apenas inventar coisas à medida que prossegue.

A Figura 7.1 mostra o esboço de um programa de agente baseado em conhecimento. Como todos os nossos agentes, ele recebe uma percepção como entrada e retorna uma ação. O agente mantém uma base de conhecimento, *KB*, que pode conter inicialmente algum **conhecimento inicial**.

função AGENTE-KB(*percepção*) **retorna** uma *ação*
  persistente: *KB*, uma base de conhecimento
    t, um contador, inicialmente igual a 0, indicando tempo
  TELL(*KB*, CRIAR-SENTENÇA-DE-PERCEPÇÃO(*percepção*, *t*))
  ação ← ASK(*KB* CRIAR-CONSULTA-DE-AÇÃO(*t*))
  TELL(*KB*, CRIAR-SENTENÇA-DE-AÇÃO(*ação*, *t*))
  t ← *t* + 1
  retornar *ação*

Figura 7.1 Um agente baseado em conhecimento genérico. Dada uma percepção, o agente adiciona a

percepção na sua base de conhecimento, pergunta à base de conhecimento qual a melhor ação e informa à base de conhecimento que executou de fato essa ação.

Cada vez que o programa do agente é chamado, ele executa duas ações. Primeiro, informa (com TELL) à base de conhecimento o que percebe. Em segundo lugar, pergunta (com ASK) à base de conhecimento que ação deve executar. No processo de responder a essa consulta, talvez seja desenvolvido um raciocínio extenso sobre o estado atual do mundo, sobre os resultados de sequências de ações possíveis, e assim por diante. Terceiro, o programa agente INFORMA (TELL) para a base de conhecimento que ação foi escolhida, e o agente executa a ação.

Os detalhes da linguagem de representação estão ocultos em duas funções que implementam a interface entre os sensores e atuadores, de um lado, e entre a representação central e o sistema de raciocínio, do outro. CRIAR-SENTENÇA-DE-PERCEPÇÃO constrói uma sentença afirmando que o agente percebeu a percepção no instante dado. CRIAR-CONSULTA-DE-AÇÃO constrói uma sentença que pergunta que ação deve ser executada nesse instante. Finalmente, CRIAR-SENTENÇA-DE-AÇÃO constrói uma sentença afirmando que a ação escolhida foi executada. Os detalhes dos mecanismos de inferência ficam ocultos dentro de TELL e ASK. As próximas seções revelarão esses detalhes.

O agente da Figura 7.1 parece bastante semelhante aos agentes com estado interno descritos no Capítulo 2. Porém, devido às definições de TELL e ASK, o agente baseado em conhecimento não é um programa arbitrário para calcular ações. Ele se adapta a uma descrição no **nível de conhecimento**, em que precisamos especificar apenas o que o agente sabe e quais são suas metas, a fim de corrigir seu comportamento. Por exemplo, um táxi automatizado poderia ter a meta de pegar um passageiro em San Francisco para Marin County e talvez soubesse que está em San Francisco e que a ponte Golden Gate é a única ligação entre os dois locais. Então, podemos esperar que ele cruze a ponte Golden Gate *porque sabe que isso o levará a atingir sua meta*. Note que essa análise é independente de como o táxi funciona no **nível de implementação**. Não importa se seu conhecimento geográfico é implementado como listas encadeadas ou mapas de pixels, ou ainda se ele raciocina manipulando strings de símbolos armazenados em registradores ou propagando sinais com ruído em uma rede de neurônios.

Um agente baseado em conhecimento pode ser construído simplesmente informando (TELLing) o que é necessário saber. Começando com uma base de conhecimento vazia, o agente projetista pode INFORMAR (TELL) sentenças uma a uma até que o agente saiba como operar em seu ambiente. Isso se chama abordagem **declarativa** para a construção de sistemas. Em contraste, a abordagem **procedural** codifica comportamentos desejados diretamente como código de programa. Nas décadas de 1970 e 1980, os defensores das duas abordagens se engajaram em acalorados debates. Agora, entendemos que um agente bem-sucedido sempre deve combinar elementos declarativos e procedurais em seu projeto e que o conhecimento declarativo muitas vezes pode ser compilado em um código procedural mais eficiente.

Podemos também fornecer a um agente baseado em conhecimento mecanismos que lhe permitam aprender por si mesmo. Esses mecanismos, descritos no Capítulo 18, criam conhecimento geral sobre o ambiente a partir de uma série de percepções. Um agente que aprende pode ser totalmente autônomo.

7.2 O MUNDO DE WUMPUS

Nesta seção, vamos descrever um ambiente em que os agentes baseados em conhecimento podem mostrar o seu valor. O **mundo de wumpus** é uma caverna que consiste em salas conectadas por passagens. À espreita em algum lugar na caverna está o terrível wumpus, um monstro que devora qualquer guerreiro que entrar em sua sala. O wumpus pode ser atingido por um agente, mas o agente só tem uma flecha. Algumas salas contêm poços sem fundo nos quais cairá qualquer um que vagar por elas (com exceção do wumpus, que é muito grande para cair em um poço). A única característica que abranda esse ambiente desolador é a possibilidade de encontrar um monte de ouro. Embora o mundo de wumpus seja bastante domesticado para os padrões dos modernos jogos de computador, ele ilustra alguns pontos importantes sobre a inteligência.

Uma amostra de mundo de wumpus é apresentada na Figura 7.2. É dada a definição precisa do ambiente de tarefa, como sugere a Seção 2.3, de acordo com a descrição de PEAS:

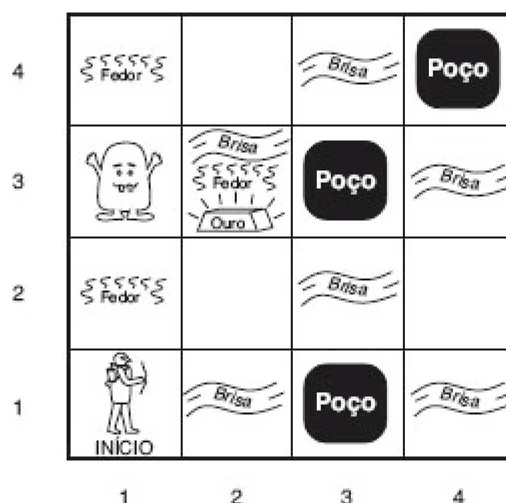


Figura 7.2 Um mundo de wumpus típico. O agente está no canto inferior esquerdo virado para a direita.

- **Medida de desempenho:** +1.000 para sair da caverna com o ouro, –1.000 se cair em um poço ou for devorado pelo wumpus, –1 para cada ação executada e –10 pelo uso da flecha. O jogo termina quando o agente morre ou quando sai da caverna.
- **Ambiente:** Uma malha 4 × 4 de salas. O agente sempre começa no quadrado identificado como [1,1], voltado para a direita. As posições do ouro e do wumpus são escolhidas ao acaso, com uma distribuição uniforme, a partir dos outros quadrados, diferentes do quadrado inicial. Além disso, cada quadrado com exceção do inicial pode ser um poço, com probabilidade 0,2.
- **Atuadores:** O agente pode mover-se *ParaFrente, VirarEsquerda* 90° ou *VirarDireita* 90°. O agente terá uma morte horrível se entrar em um quadrado contendo um poço ou um wumpus vivo (é seguro, embora fedorento, entrar em um quadrado com um wumpus morto). Se um agente tenta mover-se para a frente e esbarra em uma parede, o agente não se move. A ação *Agarrar* pode ser usada para levantar um objeto que está no mesmo quadrado em que se encontra o agente. A ação *Atirar* pode ser usada para disparar uma flecha em linha reta diante do agente. A flecha continuará até atingir (e, consequentemente, matar) o wumpus ou até atingir uma parede. O agente só tem uma flecha e, portanto, apenas a primeira ação *Atirar* tem algum efeito. Finalmente, a

ação *Escalar* pode ser usada para sair da caverna, mas apenas do quadrado [1,1].

- **Sensores:** O agente tem cinco sensores, cada um dos quais fornece um único bit de informação:
  - No quadrado contendo o wumpus e nos quadrados diretamente adjacentes (não em diagonal), o agente perceberá um *Fedor*.
  - Nos quadrados diretamente adjacentes a um poço, o agente perceberá uma *Brisa*.
  - No quadrado onde está o ouro, o agente perceberá um *Brilho*.
  - Quando caminhar para uma parede, o agente perceberá um *Impacto*.
  - Quando o wumpus é morto, ele emite um *Grito* triste que pode ser percebido em qualquer lugar na caverna.

  As percepções serão dadas ao programa do agente sob a forma de uma lista de cinco símbolos; por exemplo, se houver um fedor e uma brisa, mas nenhum brilho, impacto ou grito, o programa do agente receberá a percepção [*Fedor*, *Brisa*, *Nada*, *Nada*, *Nada*].

Podemos definir o ambiente de wumpus de acordo com as várias dimensões dadas no Capítulo 2. Claramente, é discreto, estático e de agente único (o wumpus não se move, felizmente). É sequencial, porque as recompensas podem vir somente após muitas ações serem tomadas. É parcialmente observável, porque alguns aspectos do estado não são diretamente perceptíveis: a localização do agente, o estado de saúde do wumpus e a disponibilidade de uma seta. Com relação aos locais dos poços e do wumpus: poderíamos tratá-los como as partes não observáveis do estado que são imutáveis — e, nesse caso, o modelo de transição para o ambiente é completamente conhecido ou poderíamos dizer que o modelo de transição em si é desconhecido porque o agente não sabe quais ações *ParaFrente* são fatais — em cada caso, descobrir a localização dos poços e do wumpus completa o conhecimento do agente do modelo de transição.

Para um agente no ambiente, o principal desafio é a sua ignorância inicial da configuração do ambiente; superar essa ignorância parece exigir raciocínio lógico. Na maioria das instâncias do mundo de wumpus, é possível para o agente recuperar o ouro em segurança. Ocasionalmente, o agente deve escolher entre ir para casa de mãos vazias e se arriscar a morrer para encontrar o ouro. Cerca de 21% dos ambientes são totalmente injustos porque o ouro está em um poço ou cercado por poços.

Vamos acompanhar um agente de wumpus baseado em conhecimento em sua exploração do ambiente mostrado na Figura 7.2. Usamos uma linguagem de representação do conhecimento informal que consiste em escrever símbolos em uma grade (como nas Figuras 7.3 e 7.4).

A base de conhecimento inicial do agente contém as regras do ambiente, como descrito anteriormente; em particular, ele sabe que está em [1,1] e que [1,1] é um quadrado seguro; isso está indicado com um “A” e “OK”, respectivamente, no quadrado [1,1].

A primeira percepção é [*Nada*, *Nada*, *Nada*, *Nada*, *Nada*], a partir da qual o agente pode concluir que seus quadrados vizinhos, [1,2] e [2,1], são seguros — estão OK. A Figura 7.3(a) mostra o estado de conhecimento do agente nesse momento.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

A = *Agente*
B = *Brisa*
R = *Brilho, Ouro*
OK = *Quadrado seguro*
P = *Poço*
F = *Fedor*
V = *Visitado*
W = *Wumpus*

(a)

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

(b)

Figura 7.3 O primeiro passo dado pelo agente no mundo de wumpus. (a) A situação inicial, depois da percepção [*Nada*, *Nada*, *Nada*, *Nada*, *Nada*]. (b) Depois de um movimento, com a percepção [*Nada*, *Brisa*, *Nada*, *Nada*, *Nada*].

Um agente cauteloso só se moverá para um quadrado se souber que o quadrado está OK. Vamos supor que o agente decida se mover para a frente até [2,1]. O agente detecta uma brisa (indicada por "B") em [2,1] e, assim, deve haver um poço em um quadrado vizinho. O poço não pode estar em [1,1], de acordo com as regras do jogo, e, portanto, deve haver um poço em [2,2], [3,1] ou ambos. A notação "P"? da Figura 7.3(b) indica um possível poço nesses quadrados. Nesse momento, existe apenas um quadrado conhecido que está OK e que ainda não foi visitado. Desse modo, o agente prudente se voltará, retornará a [1,1] e depois prosseguirá para [1,2].

O agente percebe um fedor em [1,2], resultando no estado de conhecimento mostrado na Figura 7.4(a). O fedor em [1,2] significa que deve haver um wumpus por perto. No entanto, o wumpus não pode estar em [1,1], pelas regras do jogo, e não pode estar em [2,2] (ou o agente teria detectado um fedor quando estava em [2,1]). Portanto, o agente pode deduzir que o wumpus está em [1,3]. A notação W! indica essa inferência. Além disso, a falta de uma brisa em [1,2] implica que não existe nenhum poço em [2,2]. O agente já inferiu que deve haver um poço em [2,2] ou [3,1], então isso significa que ele tem de estar em [3,1]. Essa inferência é bastante difícil porque combina o conhecimento obtido em diferentes instantes em diferentes lugares e se baseia na falta de uma percepção para dar um passo crucial.

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

A = *Agente*
B = *Brisa*
R = *Brilho, Ouro*
OK = *Quadrado seguro*
P = *Poço*
F = *Fedor*
V = *Visitado*
W = *Wumpus*

(a)

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(b)

Figura 7.4 Duas fases posteriores no progresso do agente. (a) Depois do terceiro movimento, com a percepção [*Fedor*, *Nada*, *Nada*, *Nada*, *Nada*]. (b) Depois do quinto movimento, com a percepção [*Fedor*, *Brisa*, *Brilho*, *Nada*, *Nada*].

O agente agora provou a si mesmo que não existe nem poço nem wumpus em [2,2], então é *OK* mover-se para lá. Não mostraremos o estado do agente baseado em conhecimento em [2,2]; apenas supomos que o agente se volta e vai para [2,3], gerando a Figura 7.4(b). Em [2,3], o agente descobre um brilho e, assim, deve agarrar o ouro e voltar para casa.

Observe que, em cada caso no qual o agente tira uma conclusão a partir das informações disponíveis, essa conclusão tem a *garantia* de ser correta se as informações disponíveis estiverem corretas. Essa é uma propriedade fundamental do raciocínio lógico. No restante deste capítulo, descreveremos como construir agentes lógicos que podem representar as informações necessárias e tirar as conclusões tais como as descritas nos parágrafos precedentes.

7.3 LÓGICA

Esta seção resume os conceitos fundamentais de representação lógica e de raciocínio. Essas ideias bonitas são independentes de qualquer das formas particulares de lógica. Portanto, adiaremos os detalhes técnicos de qualquer forma específica de lógica até a próxima seção, utilizando em vez disso o exemplo familiar de aritmética comum.

Na Seção 7.1, dissemos que as bases do conhecimento consistem em sentenças. Essas sentenças são expressas de acordo com a **sintaxe** da linguagem de representação, que especifica todas as sentenças que são bem formadas. A noção de sintaxe é bastante clara na aritmética comum: "$x + y = 4$" é uma sentença bem formada, enquanto "$x4y+ =$" não é.

Uma lógica também deve definir a **semântica** ou o significado das sentenças. A semântica define a **verdade** de cada sentença com relação a cada **mundo possível**. Por exemplo, a semântica habitual adotada pela aritmética especifica que a sentença "$x + y = 4$" é verdadeira em um mundo no qual x é 2 e y é 2, mas é falsa em um mundo em que x é 1 e y é 1. Em lógicas-padrão, toda sentença deve ser verdadeira ou falsa em cada mundo possível — não existe nenhuma posição "intermediária".[1]

Quando precisarmos ser exatos, usaremos o termo **modelo** em vez de "mundo possível". Embora mundos possíveis possam ser imaginados como ambientes (potencialmente) reais em que o agente poderia ou não estar, os modelos são abstrações matemáticas, e cada um dos quais simplesmente fixa a verdade ou falsidade de cada sentença relevante. Em termos informais, podemos pensar em um mundo possível como, por exemplo, tendo x homens e y mulheres que se sentam em torno de uma mesa para jogar *bridge*, por exemplo, e a sentença $x + y = 4$ é verdadeira quando há quatro pessoas ao todo. Formalmente, os modelos possíveis são todas as atribuições possíveis de números reais às variáveis x e y. Cada atribuição fixa a verdade de qualquer sentença da aritmética cujas variáveis são x e y. Se uma sentença α for verdadeira no modelo m, dizemos que m **satisfaz** α ou, por vezes, m **é um modelo de** α. Utilizamos a notação $M(\alpha)$ para indicar o conjunto de todos os modelos de α.

Agora, que temos uma noção de verdade, estamos prontos para conversar sobre raciocínio lógico. Ele envolve a relação de **consequência lógica** entre sentenças — a ideia de que uma sentença

decorre logicamente de outra sentença. Em notação matemática, escrevemos

$$\alpha \models \beta$$

para indicar que a sentença α tem como consequência lógica a sentença β. A definição formal de consequência lógica é: $\alpha \models \beta$ se e somente se, em todo modelo no qual α é verdadeira, β também é verdadeira. Utilizando a notação recém-apresentada, podemos escrever

$$\alpha \models \beta \text{ se e somente se } M(\alpha) \subseteq M(\beta).$$

(Observe a direção de $\subseteq$ aqui: se $\alpha \models \beta$, então α é uma afirmação *mais forte* que β: descarta *mais* mundos possíveis.) A relação de consequência lógica é familiar em aritmética; ficamos felizes com a ideia de que a sentença $x = 0$ tem como consequência lógica a sentença $xy = 0$. É óbvio que, em qualquer modelo no qual *x é zero, xy* será zero (independentemente do valor de *y*).

Podemos aplicar o mesmo tipo de análise ao exemplo de raciocínio do mundo de wumpus dado na seção anterior. Considere a situação da Figura 7.3(b): o agente detectou nada em [1,1] e uma brisa em [2,1]. Essas percepções, combinadas com o conhecimento que o agente tem das regras do mundo de wumpus constituem a BC. O agente está interessado (entre outras coisas) em saber se os quadrados adjacentes [1,2], [2,2] e [3,1] contêm poços. Cada um dos três quadrados pode ou não conter um poço e, assim (para os propósitos deste exemplo), existem $2^3 = 8$ modelos possíveis. Esses modelos são mostrados na Figura 7.5.[2]

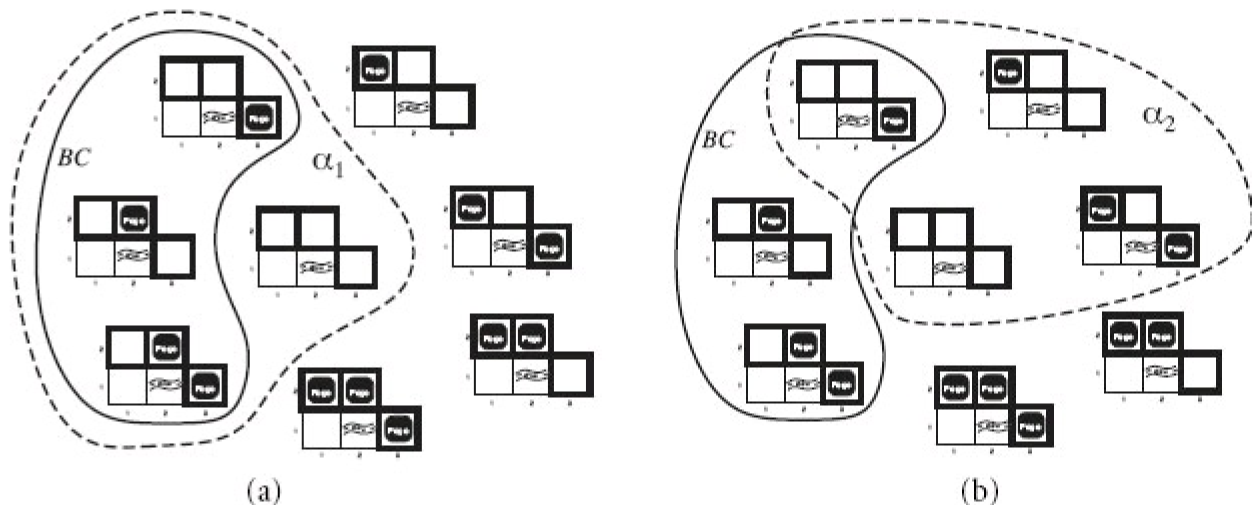


Figura 7.5 Modelos possíveis para representar a presença de poços nos quadrados [1,2], [2,2] e [3,1]. BC corresponde às observações de nada em [1,1] e brisa em [2,1] é mostrada pela linha sólida. (a) As linhas pontilhadas mostram modelos de α_1 (nenhum poço em [1,2]). (b) As linhas pontilhadas mostram modelos de α_2 (nenhum poço em [2,2]).

A BC pode ser considerada como um conjunto de sentenças ou como uma única sentença que afirma todas as sentenças individuais. A BC é falsa em modelos que contradizem o que o agente sabe — por exemplo, a BC é falsa em qualquer modelo em que [1,2] contém um poço porque não existe nenhuma brisa em [1,1]. Na verdade, há apenas três modelos em que a BC é verdadeira, e esses modelos são mostrados circundados por uma linha sólida na Figura 7.5. Agora, vamos considerar duas conclusões possíveis:

α_1 = “Não existe nenhum poço em [1,2].”
α_2 = “Não existe nenhum poço em [2,2].”

Circundamos os modelos de α_1 e α_2 com linhas pontilhadas nas Figuras 7.5(a) e 7.5(b), respectivamente. Por inspeção, vemos que:

em todo modelo no qual *BC* é verdadeira, α_1 também é verdadeira.

Consequentemente, $BC \models \alpha_1$: não existe nenhum poço em [1,2]. Também podemos ver que:

em alguns modelos nos quais *BC* é verdadeira, α_2 é falsa.

Consequentemente, $BC \not\models \alpha_2$: o agente *não pode* concluir que não existe nenhum poço em [2,2] (nem pode concluir que *existe* um poço em [2,2]).[3]

O exemplo anterior não só ilustra a consequência lógica, mas também mostra como a definição de consequência lógica pode ser aplicada para derivar conclusões, isto é, para conduzir a **inferência lógica**. O algoritmo de inferência ilustrado na Figura 7.5 é chamado **verificação de modelos** porque enumera todos os modelos possíveis para verificar que α é verdadeira em todos os modelos em que *BC* é verdadeira, isto é, que a $M(BC) \subseteq M(\alpha)$.

Na compreensão da consequência lógica e da inferência, talvez ajude pensar no conjunto de todas as consequências de *BC* como um palheiro e de a como uma agulha. A consequência lógica é como a agulha estar no palheiro; a inferência é como encontrá-la. Essa distinção está corporificada em alguma notação formal: se um algoritmo de inferência *i* pode derivar a de *BC*, escrevemos:

$$BC \vdash_i \alpha,$$

que lemos como “α é derivável de *BC* por *i*” ou “*i* deriva α de *BC*”.

O algoritmo de inferência que deriva apenas sentenças permitidas é chamado de **correto** ou se diz que ele **preserva a verdade**. A correção é uma propriedade altamente desejável. Em essência, um procedimento de inferência não correto inventa coisas à medida que prossegue — ele anuncia a descoberta de agulhas que não existem. É fácil ver que a verificação de modelos, quando aplicável,[4] é um procedimento correto.

A propriedade de **completude** também é desejável: um algoritmo de inferência será completo se puder derivar qualquer consequência lógica. No caso de palheiros reais, que têm extensão finita, parece óbvio que um exame sistemático sempre poderá definir se a agulha está no palheiro. Porém, em muitas bases de conhecimento, o palheiro de consequências é infinito, e a completude se torna uma questão importante.[5] Felizmente, existem procedimentos de inferência completos para lógicas que são suficientemente expressivas para dar conta de muitas bases de conhecimento.

☞ Descrevemos um processo de raciocínio cujas conclusões oferecem a garantia de serem verdadeiras em qualquer mundo no qual as premissas são verdadeiras; em particular, *se BC é verdadeira no* mundo real, *qualquer sentença* a *derivada de BC por um procedimento de inferência correto também será verdadeira no mundo real*. Portanto, embora um processo de inferência opere

sobre a “sintaxe” — configurações físicas internas como bits em registradores ou padrões de impulsos elétricos em cérebros — o processo *corresponde* ao relacionamento no mundo real, no qual algum aspecto do mundo real é o caso,[6] em virtude de outros aspectos do mundo real serem o caso. Essa correspondência entre o mundo e a representação está ilustrada na Figura 7.6.

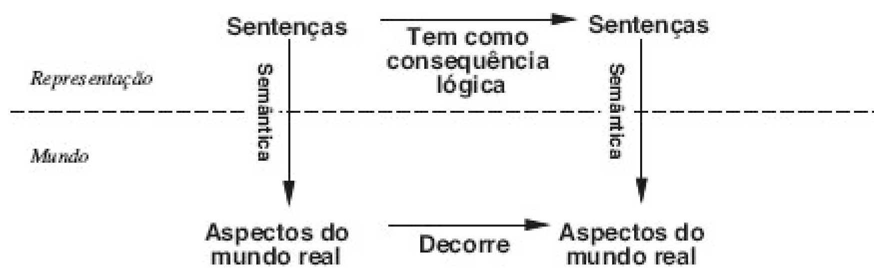


Figura 7.6 Sentenças são configurações físicas do agente, e o raciocínio é um processo de construção de novas configurações físicas a partir de configurações antigas. O raciocínio lógico deve assegurar que as novas configurações representam aspectos do mundo que na realidade decorrem dos aspectos que as antigas configurações representam.

A questão final a ser considerada é a da **fundamentação** — a conexão, se houver, entre processos de raciocínio lógico e o ambiente real em que o agente existe. Em particular, *como sabemos que a BC é verdadeira no mundo real*? (Afinal, a *BC* é apenas a “sintaxe” na cabeça do agente.) Essa é uma questão filosófica sobre a qual muitos e muitos livros foram escritos (veja o Capítulo 26). Uma resposta simples é que os sensores do agente criam a conexão. Por exemplo, nosso agente do mundo de wumpus tem um sensor de odor. O programa do agente cria uma sentença adequada sempre que há um odor. Então, sempre que a sentença está na base de conhecimento, ela é verdadeira no mundo real. Desse modo, o significado e a verdade de sentenças de percepções são definidos pelos processos de detecção e construção de sentenças que as produzem. E o restante do conhecimento do agente, tal como sua convicção de que um wumpus provoca odores em quadrados adjacentes? Essa não é uma representação direta de uma única percepção, mas uma regra geral — talvez derivada da experiência perceptiva, mas não idêntica a uma declaração dessa experiência. Regras gerais como essa são produzidas por um processo de construção de sentenças chamado **aprendizado**, que é o assunto da Parte V. O aprendizado é passível de falhas. Poderia ocorrer de um wumpus provocar cheiro, *exceto no dia* 29 *de fevereiro de anos bissexto*s, que é o dia em que eles tomam banho. Portanto, a *BC* pode não ser verdadeira no mundo real, mas há razão para otimismo no caso de bons procedimentos de aprendizado.

7.4 LÓGICA PROPOSICIONAL: UMA LÓGICA MUITO SIMPLES

Agora, apresentaremos uma lógica muito simples chamada **lógica proposicional**. Abordaremos a sintaxe da lógica proposicional e sua semântica — o modo pelo qual a verdade das sentenças é determinada. Depois, veremos a **consequência lógica** — a relação entre uma sentença e outra sentença que decorre dela — e como isso leva a um algoritmo simples para inferência lógica. É claro que tudo tem lugar no mundo de wumpus.

7.4.1 Sintaxe

A **sintaxe** da lógica proposicional define as sentenças permitidas. As **sentenças atômicas** — os elementos sintáticos indivisíveis — consistem em um único **símbolo proposicional**. Cada símbolo proposicional representa uma proposição que pode ser verdadeira ou falsa. Utilizaremos símbolos que começam com letra maiúscula e que podem conter outras letras ou subscritos, por exemplo: P, Q, R, $W_{1,3}$ e *Norte*. Os nomes são arbitrários, com frequência escolhidos de forma a apresentar algum valor mnemônico para o leitor — podemos usar $W_{1,3}$ para representar a proposição de que o wumpus está em [1,3]. (Lembre-se de que símbolos como $W_{1,3}$ são *atômicos*, isto é, W, 1 e 3 não são partes significativas do símbolo.) Existem dois símbolos proposicionais com significados fixos: *Verdadeiro* é a proposição sempre verdadeira e *Falso* é a proposição sempre falsa.

As **sentenças complexas** são construídas a partir de sentenças mais simples com a utilização de parênteses e **conectivos lógicos**. Existem cinco conectivos de uso comum:

¬ (não). Uma sentença como $\neg W_{1,3}$ é chamada **negação** de $W_{1,3}$. Um **literal** é uma sentença atômica (um **literal positivo**) ou uma sentença atômica negada (um **literal negativo**).

∧ (e). Uma sentença cujo principal conectivo é ∧, como $W_{1,3} \wedge P_{3,1}$, é chamada **conjunção**; suas partes são os elementos da **conjunção**. (O símbolo ∧ é semelhante a “A” para indicar “And” — “E” em inglês.)

∨ (ou). Uma sentença que utiliza ∨, como $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$, é uma **disjunção** dos **disjuntos** $(W_{1,3} \wedge P_{3,1})$ e $W_{2,2}$. (Historicamente, o símbolo ∨ vem da palavra latina “vel”, que significa “ou”. Para a maioria das pessoas, é mais fácil memorizá-lo como um símbolo ∧ invertido.)

⇒ (implica). Uma sentença como $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$ é chamada **implicação** (ou condicional). Sua **premissa** ou **antecedente** é $(W_{1,3} \wedge P_{3,1})$, e sua **conclusão** ou **consequente** é $\neg W_{2,2}$. As implicações também são conhecidas como **regras** ou declarações **se–então**. O símbolo de implicação às vezes é escrito em outros livros como ⊃ ou →.

⇔ (se e somente se). A sentença $W_{1,3} \Leftrightarrow \neg W_{2,2}$ é uma **bicondicional**. Alguns livros escrevem como ≡.

A Figura 7.7 apresenta uma gramática formal da lógica proposicional; consulte a página 1060 se não estiver familiarizado com a notação BNF. A gramática BNF por si só é ambígua; uma sentença com vários operadores pode ser analisada de várias maneiras pela gramática. Para eliminar a ambiguidade definimos uma precedência para cada operador. O operador “não” (¬) tem a precedência mais alta, o que significa que na sentença $\neg A \wedge B$ o ¬ liga-se mais firmemente, dando-nos o equivalente de $(\neg A) \wedge B$ em vez de $\neg(A \wedge B)$ (a notação para a aritmética comum é a mesma: $-2 + 4$ são 2 e não -6). Em caso de dúvida, utilize parênteses para certificar-se da interpretação correta. Colchetes quadrados significam a mesma coisa que parênteses; o objetivo da escolha de colchetes quadrados ou parênteses é apenas tornar mais fácil a leitura da sentença por um ser humano.

Sentença → *SentençaAtômica* | *SentençaComplexa*
SentençaAtômica → *Verdadeiro* | *Falso* | *P* | *Q* | *R* | ...
SentençaComplexa → (*Sentença*) | [*Sentença*]
| ¬ *Sentença*
| (*Sentença* ∧ *Sentença*)
| (*Sentença* ∨ *Sentença*)
| (*Sentença* ⇒ *Sentença*)
| (*Sentença* ⇔ *Sentença*)

PRECEDÊNCIA DE OPERADORES : ¬, ∧, ∨, ⇒, ⇔

Figura 7.7 Uma gramática BNF (Backus–Naur Form) de sentenças em lógica proposicional, com precedências de operadores do mais alto para o mais baixo.

7.4.2 Semântica

Tendo especificado a sintaxe da lógica proposicional, agora especificaremos sua semântica. A semântica define as regras para determinar a verdade de uma sentença com respeito a um modelo específico. Em lógica proposicional, um modelo simplesmente fixa o **valor-verdade** — *verdadeiro* ou *falso* — para todo símbolo proposicional.

Por exemplo, se as sentenças na base de conhecimento fizerem uso dos símbolos proposicionais $P_{1,2}$, $P_{2,2}$ e $P_{3,1}$, um modelo possível será:

$$m_1 = \{P_{1,2} = falsa, P_{2,2} = falsa, P_{3,1} = verdadeira\}.$$

Com três símbolos proposicionais, existem $2^3 = 8$ modelos possíveis — exatamente aqueles que estão representados na Figura 7.5. Porém, note que os modelos são objetos puramente matemáticos, sem qualquer conexão necessária para mundos de wumpus. $P_{1,2}$ é apenas um símbolo; ele poderia significar "existe um poço em [1,2]" ou "estarei em Paris hoje e amanhã".

A semântica da lógica proposicional deve especificar como calcular o valor verdade de *qualquer* sentença, dado um modelo. Isso é feito de forma recursiva. Todas as sentenças são construídas a partir de sentenças atômicas e dos cinco conectivos; assim, precisamos especificar como calcular a verdade de sentenças atômicas e como calcular a verdade de sentenças formadas com cada um dos cinco conectivos. As sentenças atômicas são fáceis:

- *Verdadeiro* é verdadeiro em todo modelo e *Falso* é falso em todo modelo.
- O valor-verdade de todos os outros símbolos proposicionais deve ser especificado diretamente no modelo. Por exemplo, no modelo m_1 dado anteriormente, $P_{1,2}$ é falsa.

Para sentenças complexas, temos cinco regras, que valem para quaisquer subsentenças P e Q em qualquer modelo m (aqui "sse" significa "se e somente se"):

- $\neg P$ é verdadeiro sse P for falso em m.
- $P \land Q$ são verdadeiros sse P e Q forem verdadeiros em m.

- $P \lor Q$ é verdadeiro sse P ou Q for verdadeiro em m.
- $P \Rightarrow Q$ é verdadeiro, a menos que P seja verdadeiro e Q seja falso em m.
- $P \Leftrightarrow Q$ é verdadeiro sse P e Q forem ambos verdadeiros ou ambos falsos em m.

As regras também podem ser expressas em **tabelas-verdade** que especificam o valor-verdade de uma sentença complexa para cada atribuição possível de valores-verdade a seus componentes. As tabelas-verdade para os cinco conectivos lógicos são dadas na Figura 7.8. Dessas tabelas, o valor-verdade de qualquer sentença s pode ser calculado com relação a qualquer modelo m por um processo simples de avaliação recursiva. Por exemplo, a sentença $\neg P_{1,2} \land (P_{2,2} \lor P_{3,1})$, avaliada em m_1, resulta em *verdadeiro* ∧ (*falso* ∨ *verdadeiro*) = *verdadeiro* ∧ *verdadeiro* = *verdadeiro*. O Exercício 7.3 pede para escrever o algoritmo VERDADEIRO-LP?(s, m), que calcula o valor-verdade de uma sentença de lógica proposicional s em um modelo m.

P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
falso	*falso*	*verdadeiro*	*falso*	*falso*	*verdadeiro*	*verdadeiro*
falso	*verdadeiro*	*verdadeiro*	*falso*	*verdadeiro*	*verdadeiro*	*falso*
verdadeiro	*falso*	*falso*	*falso*	*verdadeiro*	*falso*	*falso*
verdadeiro	*verdadeiro*	*falso*	*verdadeiro*	*verdadeiro*	*verdadeiro*	*verdadeiro*

Figura 7.8 Tabelas-verdade referentes aos cinco conectivos lógicos. Se quiser usar a tabela para calcular, por exemplo, o valor de $P \lor Q$ quando P é verdadeira e Q é falsa, primeiro procure no lado esquerdo a linha em que P é *verdadeira* e Q é *falsa* (a terceira linha). Em seguida, observe nessa linha sob a coluna $P \lor Q$ o resultado: *verdadeiro*.

A tabelas-verdade para "e", "ou" e "não" estão intimamente relacionadas às nossas intuições sobre as palavras em nosso idioma. O principal ponto de confusão possível é que $P \lor Q$ é verdadeira quando P é verdadeira ou Q é verdadeira, *ou ambas*. Existe um conectivo diferente chamado "ou exclusivo" ("xor" para abreviar) que resulta em falso quando ambos os disjuntos são verdadeiros.[7] Não há nenhum consenso quanto ao símbolo para ou exclusivo; algumas opções são ∨ ou ≠ ou ⊕.

A tabela-verdade para) ⇒ pode parecer enigmática à primeira vista, porque talvez não se encaixe muito bem na compreensão intuitiva de que "P implica Q" ou "se P então Q". Em primeiro lugar, a lógica proposicional não exige qualquer relação de causa e efeito ou relevância entre P e Q. A sentença "5 é ímpar implica que Tóquio é a capital do Japão" é uma sentença verdadeira da lógica proposicional (sob a interpretação normal), embora seja uma sentença decididamente estranha em português. Outro ponto de confusão é que qualquer implicação é verdadeira sempre que sua antecedente é falsa. Por exemplo, "5 é par implica que Sam é inteligente" é verdadeira, independentemente de Sam ser ou não inteligente. Isso parece estranho, mas faz sentido se você pensar em "$P \Rightarrow Q$" como "se P é verdadeira, então estou afirmando que Q é verdadeira. Caso contrário, não estou fazendo nenhuma afirmação". O único modo de essa sentença ser *falsa* é P ser verdadeira, mas Q ser falsa.

A bicondicional $P \Leftrightarrow Q$ é verdadeira sempre que tanto $P \Rightarrow Q$ quanto $Q \Rightarrow P$ sejam verdadeiras.

Em nosso idioma, isso frequentemente é escrito como "*P* se e somente se *Q*". Muitas das regras do mundo de wumpus são mais bem escritas usando-se ⇔. Por exemplo, um quadrado tem uma brisa *se* um quadrado vizinho tem um poço, e um quadrado tem uma brisa *somente se* um quadrado vizinho tem um poço. Assim, precisamos de bicondicionais como:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}),$$

onde $B_{1,1}$ significa que existe uma brisa em [1,1].

7.4.3 Uma base de conhecimento simples

Agora que definimos a semântica para a lógica proposicional, podemos construir uma base de conhecimento para o mundo de wumpus. Vamos concentrar-nos primeiro nos aspectos *imutáveis* do mundo de wumpus, deixando os aspectos mutáveis para uma seção posterior. Por enquanto, precisamos dos seguintes símbolos para cada localização $[x, y]$:

$P_{x,y}$ é verdadeiro se existe um poço em $[x, y]$.

$W_{x,y}$ é verdadeiro se existe um wumpus em $[x, y]$, vivo ou morto.

$B_{x,y}$ é verdadeiro se o agente percebe uma brisa em $[x, y]$.

$S_{x,y}$ é verdadeiro se o agente percebe um fedor em $[x, y]$.

As sentenças que escrevemos serão suficientes para derivar $\neg P_{1,2}$ (não há poço em [1,2]), como feito informalmente na Seção 7.3. Rotularemos cada sentença de R_i, para que possamos nos referir a elas:

- Não há nenhum poço em [1,1]:

$$R_1: \neg P_{1,1}.$$

- Um quadrado tem uma brisa se e somente se existe um poço em um quadrado vizinho. Isso tem de ser declarado para cada quadrado; por enquanto, incluímos apenas os quadrados relevantes:

$$R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}).$$
$$R_3: B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}).$$

- As sentenças precedentes são verdadeiras em todos os mundos de wumpus. Agora, incluímos as percepções de brisa para os dois primeiros quadrados visitados no mundo específico em que o agente se encontra, levando à situação da Figura 7.3(b).

$$R_4: \neg B_{1,1}.$$
$$R_5: B_{2,1}.$$

7.4.4 Um procedimento de inferência simples

Nosso objetivo agora é decidir se $BC \models \alpha$ para alguma sentença α. Por exemplo, $\neg P_{1,2}$ é consequência lógica da nossa BC? Nosso primeiro algoritmo para inferência é uma abordagem de um modelo de verificação que é uma implementação direta da definição de consequência lógica: enumere os modelos e verifique se α é verdadeira em todo modelo no qual *BC* é verdadeira. No caso da lógica proposicional, os modelos são atribuições de *verdadeiro* ou *falso* a todo símbolo proposicional.

Voltando ao nosso exemplo do mundo de wumpus, os símbolos proposicionais relevantes são $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$ e $P_{3,1}$. Com sete símbolos, existem $2^7 = 128$ modelos possíveis; em três desses modelos, *BC* é verdadeira (Figura 7.9). Nesses três modelos, $\neg P_{1,2}$ é verdadeira; consequentemente, não existe nenhum poço em [1,2]. Por outro lado, $P_{2,2}$ é verdadeira em dois dos três modelos e falsa em um, e assim não podemos dizer ainda se existe um poço em [2,2].

A Figura 7.9 reproduz de forma mais precisa o raciocínio ilustrado na Figura 7.5. Um algoritmo geral para decidir a consequência lógica em lógica proposicional é mostrado na Figura 7.10. Como no algoritmo BUSCA-COM-RETROCESSO na página 186, CONSEQUÊNCIA-LÓGICA? executa uma enumeração recursiva de um espaço finito de atribuições a variáveis. O algoritmo é **consistente**, porque implementa diretamente a definição de consequência lógica, e é **completo**, porque funciona para qualquer *BC* e a, e sempre termina — só existe um número finito de modelos a examinar.

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1
falso	*falso*	*falso*	*falso*	*falso*	*falso*	*falso*	*verdad*
falso	*falso*	*falso*	*falso*	*falso*	*falso*	*verdadeiro*	*verdad*
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
falso	*verdadeiro*	*falso*	*falso*	*falso*	*falso*	*falso*	*verdad*
falso	*verdadeiro*	*falso*	*falso*	*falso*	*falso*	*verdadeiro*	*verdad*
falso	*verdadeiro*	*falso*	*falso*	*falso*	*verdadeiro*	*falso*	*verdad*
falso	*verdadeiro*	*falso*	*falso*	*falso*	*verdadeiro*	*verdadeiro*	*verdad*
falso	*verdadeiro*	*falso*	*falso*	*verdadeiro*	*falso*	*falso*	*verdad*
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
verdadeiro	*verdadeiro*	*verdadeiro*	*verdadeiro*	*verdadeiro*	*verdadeiro*	*verdadeiro*	*falso*

Figura 7.9 Uma tabela-verdade construída para a base de conhecimento dada no texto. *BC* é verdadeira se *R*1 a *R*5 são verdadeiras, o que acontece em apenas três das 128 linhas (as que estão sublinhados na coluna do lado direito). Em todas as três linhas, $P_{1,2}$ é falsa e, assim, não existe nenhum poço em [1,2]. Por outro lado, pode haver (ou não) um poço em [2,2].

```
função CONSEQUÊNCIA-LÓGICA-TV?(BC, α) devolve verdadeiro ou falso
   entradas: BC, a base de conhecimento, uma sentença em lógica proposicional
             α, a consulta, uma sentença em lógica proposicional

   símbolos ← uma lista dos símbolos proposicionais em BC e α
   devolver VERIFICAR-TODOS-TV(BC, α, símbolos, { })
```

```
função VERIFICAR-TODOS-TV(BC, α, símbolos, modelo) devolve verdadeiro ou falso
   se VAZIO?(símbolos) então
      se VERDADEIRO-LP?(BC, modelo) então devolver VERDADEIRO-LP?(α, modelo)
      senão devolver verdadeiro // quando BC for falso, sempre retornar verdadeiro
   senão faça
      P ← PRIMEIRO(símbolos)
      restante ← RESTO(símbolos)
      devolver VERIFICAR-TODOS-TV(BC, α, restante, modelo ∪ {P = verdadeiro})
             e
             VERIFICAR-TODOS-TV(BC, α, restante, modelo ∪ {P = falso})
```

Figura 7.10 Um algoritmo de enumeração de tabela-verdade para decidir a consequência lógica proposicional. (TV significa tabela-verdade.) VERDADEIRO-LP? retorna *verdadeiro* se uma sentença é válida dentro de um modelo. A variável *modelo* representa um modelo parcial — uma atribuição a alguns dos símbolos. A palavra-chave "**e**" é aqui utilizada como uma operação lógica sobre seus dois argumentos, retornando *verdadeiro* ou *falso*.

É claro que "um número finito" nem sempre é o mesmo que "poucos". Se *BC* e a contêm *n* símbolos ao todo, então existem 2^n modelos. Desse modo, a complexidade de tempo do algoritmo é $O(2^n)$. (A complexidade de espaço é somente $O(n)$ porque a enumeração é feita em profundidade.) Mais adiante neste capítulo, veremos algoritmos que são muito mais eficientes na prática. Infelizmente, a consequência lógica proposicional é co-NP-completa (ou seja, provavelmente não mais fácil que NP-completa — veja o Apêndice A); assim, *todo algoritmo de inferência conhecido para lógica proposicional tem uma complexidade no pior caso que é exponencial em relação ao tamanho da entrada*.

7.5 PROVA DE TEOREMAS PROPOSICIONAIS

Até agora, mostramos como determinar a consequência lógica por *verificação de modelos*: enumerar modelos e mostrar que a sentença deve valer em todos os modelos. Nesta seção, mostraremos como se pode determinar a consequência lógica através da **prova de teoremas** — com a aplicação direta de regras de inferência nas sentenças em nossa base de conhecimento para construir uma prova da sentença desejada sem consultar os modelos. Se o número de modelos for

grande mas o comprimento da prova for curto, a demonstração do teorema pode ser mais eficiente do que a verificação de modelos.

Antes de mergulhar nos detalhes dos algoritmos de prova de teoremas, vamos precisar de alguns conceitos adicionais relacionados à consequência lógica.

O primeiro conceito é o de **equivalência lógica**: duas sentenças α e β são logicamente equivalentes se são verdadeiras no mesmo conjunto de modelos. Escrevemos isso como $\alpha \equiv \beta$. Por exemplo, podemos mostrar facilmente (utilizando tabelas-verdade) que $P \land Q$ e $Q \land P$ são logicamente equivalentes; outras equivalências são mostradas na Figura 7.11. Essas equivalências desempenham o mesmo papel em lógica que as identidades aritméticas desempenham na matemática comum. Uma definição alternativa de equivalência é: duas sentenças quaisquer α e β são equivalentes apenas se cada uma delas for a consequência lógica da outra:

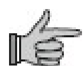
$(\alpha \land \beta) \equiv (\beta \land \alpha)$ comutatividade de $\land$
$(\alpha \lor \beta) \equiv (\beta \lor \alpha)$ comutatividade de $\lor$
$((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma))$ associatividade de $\land$
$((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma))$ associatividade de $\lor$
$\neg(\neg\alpha) \equiv \alpha$ eliminação da dupla negação
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ contraposição
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \lor \beta)$ eliminação da implicação
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$ eliminação da bicondicional
$\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta)$ De Morgan
$\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta)$ De Morgan
$(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))$ distribuição de $\land$ sobre $\lor$
$(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))$ distribuição de $\lor$ sobre $\land$

Figura 7.11 Equivalências lógicas comuns. Os símbolos α, β e γ representam sentenças arbitrárias de lógica proposicional.

$\alpha \equiv \beta$, se e somente se $\alpha \models \beta$ e $\beta \models \alpha$.

O segundo conceito de que precisaremos é o de **validade**. Uma sentença é válida se é verdadeira em *todos* os modelos. Por exemplo, a sentença $P \lor \neg P$ é válida. As sentenças válidas também são conhecidas como **tautologias** — elas são *necessariamente* verdadeiras. Como a sentença *Verdadeiro* é verdadeira em todos os modelos, toda sentença válida é logicamente equivalente a *Verdadeiro*.

Qual é a utilidade das sentenças válidas? De nossa definição de consequência lógica, podemos derivar o **teorema da dedução**, conhecido pelos gregos antigos:

 *Para quaisquer sentenças α e β, $\alpha \models \beta$ se e somente se a sentença $\alpha \Rightarrow \beta$ é válida.*

(O Exercício 7.5 lhe pede uma prova.) Assim, podemos decidir se $\alpha \models \beta$ verificando que $(\alpha \Rightarrow \beta)$ é verdadeiro em cada modelo — o que é essencialmente o que o algoritmo de inferência na Figura 7.10

faz — ou provando que ($\alpha \Rightarrow \beta$) equivale a *Verdadeiro*. Por outro lado, o teorema da dedução determina que toda sentença de implicação válida descreve uma inferência legítima.

O último conceito de que precisaremos é o de **satisfatibilidade**. Uma sentença é satisfatível se for verdadeira em ou satisfeita por *algum* modelo. Por exemplo, a base de conhecimento dada anteriormente, ($R_1 \land R_2 \land R_3 \land R_4 \land R_5$), é satisfatível porque existem três modelos em que ela é verdadeira, como mostra a Figura 7.9. A satisfatibilidade pode ser verificada enumerando-se os modelos possíveis até ser encontrado um modelo que satisfaça a sentença. O problema de determinar a satisfatibilidade de sentenças em lógica proposicional — o problema **SAT** — foi o primeiro problema que se comprovou ser NP-completo. Muitos problemas em ciência da computação na verdade são problemas de satisfatibilidade. Por exemplo, todos os problemas de satisfação de restrições do Capítulo 6 estão essencialmente perguntando se as restrições são satisfatíveis por alguma atribuição.

É claro que a validade e a satisfatibilidade estão interconectadas: α é válida se e somente se ¬a não é satisfatível; em contraposição, a é satisfatível se e somente se ¬a não é válida. Também temos o seguinte resultado útil:

$\alpha \models \beta$ se e somente se a sentença ($\alpha \land \neg\beta$) é não-satisfatível.

Provar β a partir de α pela verificação da não satisfatibilidade de ($\alpha \land \neg\beta$) corresponde exatamente à técnica-padrão de prova matemática de *reductio ad absurdum* (literalmente, "redução ao absurdo"). Ela também é chamada prova por **refutação** ou prova por **contradição**. Supõe-se que uma sentença β seja falsa e se demonstra que ela leva a uma contradição com axiomas conhecidos α. Essa contradição é exatamente o que se quer dizer quando se afirma que a sentença ($\alpha \land \neg\beta$) é não satisfatível.

7.5.1 Inferência e provas

Esta seção abrange **regras de inferência** que podem ser aplicadas para derivar uma **prova** — uma cadeia de conclusões que conduzem ao objetivo desejado. A regra mais conhecida é chamada ***Modus Ponens*** (do Latim *modo que afirma*) e é escrita desta forma:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

A notação significa que, sempre que quaisquer sentenças da forma $\alpha \Rightarrow \beta$ e α são dadas, a sentença β pode ser deduzida. Por exemplo, se (*WumpusAdiante* $\land$ *WumpusVivo*)) $\Rightarrow$ *Atirar* e (*WumpusAdiante* $\land$ *WumpusVivo*) são dadas, *Atirar* pode ser deduzida.

Outra regra de inferência útil é **Eliminação do E**; a regra que afirma que, a partir de uma conjunção, qualquer um dos elementos da conjunção pode ser deduzido:

$$\frac{\alpha \land \beta}{\alpha}$$

Por exemplo, a partir de (*WumpusAdiante* ∧ *WumpusVivo*), *WumpusVivo* pode ser deduzido.

Considerando os valores-verdade possíveis de α e β, pode-se mostrar facilmente que *Modus Ponens* e Eliminação do E são corretas de uma vez por todas. Essas regras podem então ser usadas em quaisquer instâncias específicas a que se apliquem, gerando inferências corretas sem a necessidade de enumeração de modelos.

Todas as equivalências lógicas da Figura 7.11 podem ser usadas como regras de inferência. Por exemplo, a equivalência para a eliminação de bicondicional gera as duas regras de inferência:

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{e} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}.$$

Nem todas as regras de inferência funcionam em ambos os sentidos como essa. Por exemplo, não podemos executar *Modus Ponens* no sentido oposto para obter $\alpha \Rightarrow \beta$ e α a partir de β.

Vejamos como essas regras de inferência e equivalências podem ser usadas no mundo de wumpus. Começamos com a base de conhecimento contendo R_1 a R_5 e mostramos como provar $\neg P_{1,2}$, isto é, não existe nenhum poço em [1,2]. Primeiro, aplicamos a eliminação de bicondicional a R_2 para obter:

$$R_6: \ (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}).$$

Em seguida, aplicamos a Eliminação do E a R_6 para obter:

$$R_7: \ ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}).$$

A equivalência lógica para contraposição fornece:

$$R_8: \ (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})).$$

Agora, podemos aplicar *Modus Ponens* com R^8 e a percepção R_4 (isto é, $\neg B_{1,1}$), para obter:

$$R_9: \ \neg(P_{1,2} \vee P_{2,1}).$$

Finalmente, aplicamos a regra de De Morgan, gerando a conclusão:

$$R_{10}: \ \neg P_{1,2} \wedge \neg P_{2,1}.$$

Ou seja, nem [1,2] nem [2,1] contêm um poço.

Encontramos essa prova manualmente, mas podemos aplicar qualquer um dos algoritmos de busca do Capítulo 3 para encontrar uma sequência de passos que constitui uma prova. Nós apenas precisamos definir um problema de prova da seguinte forma:

- ESTADO INICIAL: a base de conhecimento inicial.
- AÇÕES: o conjunto de ações consiste em todas as regras de inferência aplicadas a todas as sentenças que correspondam à metade superior da regra de inferência.
- RESULTADO: o resultado de uma ação é acrescentar a sentença na metade inferior da regra de

inferência.

- OBJETIVO: o objetivo é um estado que contém a sentença que estamos tentando provar.

Assim, a busca de provas é uma alternativa para a enumeração de modelos. Porém, em muitos casos práticos, *encontrar uma prova pode ser mais eficiente porque a prova pode ignorar proposições irrelevantes*, *independentemente de quantas dessas proposições existem*. Por exemplo, a prova dada anteriormente que leva a $\neg P_{1,2} \wedge \neg P_{2,1}$ não menciona as proposições $B_{2,1}$, $P_{1,1}$, $P_{2,2}$ ou $P_{3,1}$. Elas podem ser ignoradas porque a proposição objetivo $P_{1,2}$ só aparece na sentença R_2; as outras proposições em R_2 só aparecem em R_4 e em R_2; assim, R_1, R_3 e R_5 não têm nenhuma influência sobre a prova. O mesmo seria válido ainda que acrescentássemos mais um milhão de sentenças à base de conhecimento; por outro lado, o algoritmo simples de tabela-verdade seria subjugado pela explosão exponencial de modelos.

Uma propriedade final de sistemas lógicos é a **monotonicidade**, que afirma que o conjunto de consequências lógicas permitidas só pode *aumentar* à medida que as informações são acrescentadas à base de conhecimento.[8] Para quaisquer sentenças α e β,

$$\text{se } KB \models \alpha \text{ então } KB \wedge \beta \models \alpha.$$

Por exemplo, suponha que a base de conhecimento contenha a asserção adicional β afirmando que existem exatamente oito poços no mundo. Esse conhecimento poderia ajudar o agente a tirar conclusões *adicionais*, mas não pode invalidar qualquer conclusão α já deduzida — como a conclusão de que não existe nenhum poço em [1,2]. A monotonicidade significa que as regras de inferência podem ser aplicadas sempre que são encontradas premissas adequadas na base de conhecimento — a conclusão da regra deve se seguir *independentemente de outras informações existentes na base de conhecimento*.

7.5.2 Prova por resolução

Afirmamos que as regras de inferência focalizadas até agora são *corretas*, mas não discutimos a questão de *completude* para os algoritmos de inferência que as utilizam. Os algoritmos de busca como a busca por aprofundamento iterativo são completos no sentido de que encontrarão qualquer objetivo acessível; porém, se as regras de inferência disponíveis forem inadequadas, a meta não será acessível — não existirá nenhuma prova que utilize apenas essas regras de inferência. Por exemplo, se removêssemos a regra de eliminação de bicondicional, a prova apresentada na seção anterior não seria possível. A seção corrente introduz uma regra de inferência única, a **resolução**, que gera um algoritmo de inferência completo quando acoplada a qualquer algoritmo de busca completo.

Começaremos usando uma versão simples da regra de resolução no mundo de wumpus. Vamos considerar as etapas que conduzem à Figura 7.4(a): o agente retorna de [2,1] para [1,1] e vai para [1,2], onde percebe um fedor, mas nenhuma brisa. Adicionamos os fatos a seguir à base de conhecimento:

$$R_{11}: \neg B_{1,2}.$$
$$R_{12}: B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3}).$$

Pelo mesmo processo que levou a $R10$ anteriormente, agora podemos derivar a ausência de poços em [2,2] e [1,3] (lembre-se de que já sabemos que [1,1] não tem poços):

$$R_{13}: \neg P_{2,2}.$$
$$R_{14}: \neg P_{1,3}.$$

Também podemos aplicar a eliminação de bicondicional a R_3, seguida por *Modus Ponens* com R_5, a fim de obter o fato de que existe um poço em [1,1], [2,2] ou [3,1]:

$$R_{15}: P_{1,1} \vee P_{2,2} \vee P_{3,1}.$$

Agora, vem a primeira aplicação da regra de resolução: o literal $\neg P_{2,2}$ em R_{13} *se resolve com* o literal $P_{2,2}$ em R_{15} para fornecer o **resolvente**:

$$R_{16}: P_{1,1} \vee P_{3,1}.$$

Em linguagem comum: se existe um poço em [1,1], [2,2] ou [3,1] e não existe poço em [2,2], então ele está em [1,1] ou [3,1]. De modo semelhante, o literal $\neg P_{1,1}$ em R_1 se resolve com o literal $P_{1,1}$ em R_{16} para gerar:

$$R_{17}: P_{3,1}.$$

Em linguagem comum: se existe um poço em [1,1] ou [3,1] e ele não está em [1,1], então está em [3,1]. Essas duas últimas etapas de inferência são exemplos da regra de inferência de **resolução unitária**,

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k},$$

onde cada l é um literal, e onde l_i e m são **literais complementares** (isto é, um é a negação do outro). Desse modo, a regra de resolução unitária toma uma **cláusula** — uma disjunção de literais — e um literal, e produz uma nova cláusula. Observe que um único literal pode ser visto como uma disjunção de um único literal, também conhecida como **cláusula unitária**.

A regra de **resolução** unitária pode ser generalizada para a regra de **resolução** completa,

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n},$$

onde l_i e m_j são literais complementares. Isso quer dizer que a resolução recebe duas cláusulas e produz uma nova cláusula contendo todos os literais das duas cláusulas originais, *exceto* os dois literais complementares. Por exemplo, temos:

$$\frac{P_{1,1} \vee P_{3,1}, \qquad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}.$$

Existe ainda outro aspecto técnico da regra de resolução: a cláusula resultante deve conter apenas uma cópia de cada literal.[9] A remoção de várias cópias de literais é chamada **fatoração**. Por

exemplo, se resolvermos ($A \lor B$) com ($A \lor \neg B$), obteremos ($A \lor A$), que será reduzido simplesmente a A.

A *correção* da regra de resolução pode ser vista facilmente considerando-se o literal l_i que é complementar à literal m_j na outra cláusula. Se l_i é verdadeiro, então m_j é falso e, consequentemente, $m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n$ tem de ser verdadeira, porque $m_1 \lor \ldots \lor m_n$ é dada. Se l_i é falso, então $l_1 \lor \ldots \lor l_{j-1} \lor l_{i+1} \lor \ldots \lor l_k$ deve ser verdadeira porque $l_i \lor \ldots \lor l_k$ é dada. Agora, l_i é verdadeiro ou falso, e então uma ou outra dessas conclusões é válida — exatamente como estabelece a regra de resolução.

O fato mais surpreendente sobre a regra de resolução é que ela forma a base para uma família de procedimentos de inferência *completos*. *Um demonstrador de teoremas baseado em resolução pode, para quaisquer sentenças α e β em lógica proposicional, decidir se α |= β*. As duas subseções a seguir explicam como a resolução consegue isso.

Forma normal conjuntiva

A regra de resolução se aplica apenas às cláusulas (isto é, às disjunções de literais); assim, ela aparentemente só seria relevante para bases de conhecimento e consultas que consistissem em tais disjunções. Então, como ela pode levar a um procedimento de inferência completo para toda a lógica proposicional? A resposta é que *toda sentença da lógica proposicional é logicamente equivalente a uma conjunção de cláusulas*. Dizemos que uma sentença expressa como uma conjunção de cláusulas está em **forma normal conjuntiva** ou **FNC** (veja a Figura 7.14). Descreveremos agora um procedimento para converter para FNC.

Ilustraremos o procedimento convertendo a sentença $B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$ para FNC. As etapas são:

1. Eliminar ⇔, substituindo $\alpha \Leftrightarrow \beta$ por $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$:

$$(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1}).$$

2. Eliminar ⇒, substituindo $\alpha \Rightarrow \beta$ por $\neg\alpha \lor \beta$:

$$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1}).$$

3. A FNC exige que ¬ apareça apenas em literais; portanto, "movemos ¬ para dentro" pela aplicação repetida das seguintes equivalências da Figura 7.11:

$$\neg(\neg\alpha) \equiv \alpha \text{ (eliminação de negação dupla)}$$
$$\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta) \text{ (De Morgan)}$$
$$\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta) \text{ (De Morgan)}$$

No exemplo, necessitamos apenas de uma aplicação da última regra:

$$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1}).$$

4. Agora, temos uma sentença que contém operadores ∧ e ∨ aninhados, aplicados a literais. Aplicamos a lei de distributividade da Figura 7.11, distribuindo ∨ sobre ∧ sempre que possível.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}).$$

Agora, a sentença original está em FNC, como uma conjunção de três cláusulas. É muito mais difícil ler essa sentença, mas ela pode ser usada como entrada para um procedimento de resolução.

Um algoritmo de resolução

Os procedimentos de inferência baseados na resolução funcionam pela utilização do princípio de prova por contradição introduzido na pág. 218. Isto é, para mostrar que $BC \models \alpha$ mostramos que ($BC \wedge \neg\alpha$) é não satisfatível. Fazemos isso provando uma contradição.

Um algoritmo de resolução é mostrado na Figura 7.12. Primeiro, ($BC \wedge \neg\alpha$) é convertido em FNC. Em seguida, a regra de resolução é aplicada às cláusulas resultantes. Cada par que contém literais complementares é resolvido para gerar uma nova cláusula, que é adicionada ao conjunto se ainda não estiver presente. O processo continua até acontecer um destes dois fatos:

função RESOLUÇÃO-LP(*BC*, a) **retorna** *verdadeiro* ou *falso*
  entradas: *BC*, a base de conhecimento, uma sentença em lógica proposicional
        a, a consulta, uma sentença em lógica proposicional

  cláusulas ← o conjunto de cláusulas na representação de FNC de *BC* ∧ ¬a
  novas ← { }
  repita
    para cada par de cláusulas C_i, C_j **em** *cláusulas* **faça**
      resolventes ← RESOLVER-LP(C_i, C_j)
      se *resolventes* contém a cláusula vazia **então retornar** *verdadeiro*
      novas ← *novas* ∪ *resolventes*
    se *novas* ⊆ *cláusulas* **então retornar** *falso*
    cláusulas ← *cláusulas* ∪ *novas*

Figura 7.12 Um algoritmo de resolução simples para lógica proposicional. A função RESOLVER-LP retorna o conjunto de todas as cláusulas possíveis obtidas pela resolução de suas duas entradas.

- não há nenhuma cláusula nova que possa ser adicionada, nesse caso BC não tem a como consequência lógica; ou,
- duas cláusulas resolvem produzindo uma cláusula *vazia*, nesse caso BC tem como consequência lógica a.

A cláusula vazia — uma disjunção de nenhum disjunto — é equivalente a *Falso* porque uma disjunção só é verdadeira se pelo menos um de seus disjuntos é verdadeiro. Outra maneira de ver que uma cláusula vazia representa uma contradição é observar que ela só surge da solução de duas

cláusulas unitárias complementares como P e $\neg P$.

Podemos aplicar o procedimento de resolução a uma inferência muito simples no mundo de wumpus. Quando o agente está em [1,1], não existe nenhuma brisa e, assim, não pode haver poços em quadrados vizinhos. A base de conhecimento relevante é:

$$BC = R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

e desejamos provar a que é, digamos, $\neg P_{1,2}$. Quando convertermos ($BC \wedge \neg$a) em FNC, obteremos as cláusulas mostradas na parte superior da Figura 7.13. A segunda linha da figura mostra todas as cláusulas obtidas pela resolução de pares na primeira linha. Então, quando $P_{1,2}$ é resolvida com $\neg P_{1,2}$, obtemos a cláusula vazia, mostrada como um quadrado pequeno. A inspeção da Figura 7.13 revela que muitas etapas de resolução não têm sentido. Por exemplo, a cláusula $B_{1,1} \vee \neg B_{1,1} \vee P_{1,2}$ é equivalente a *Verdadeiro* $\vee$ $P_{1,2}$, que é equivalente a *Verdadeiro*. A dedução de que *Verdadeiro* é verdadeira não é muito útil. Portanto, qualquer cláusula em que aparecem dois literais complementares pode ser descartada.

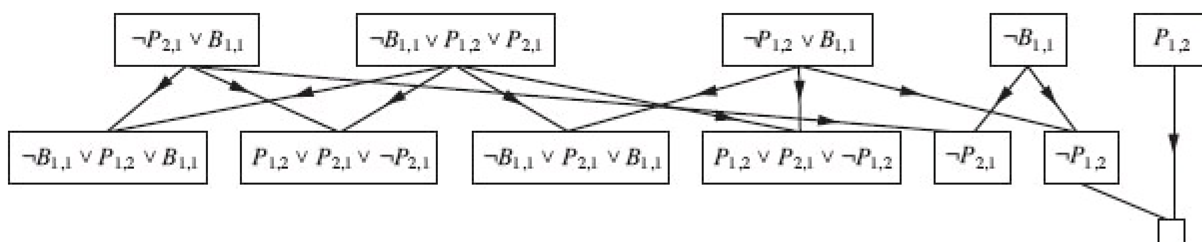


Figura 7.13 Aplicação parcial de RESOLUÇÃO-LP a uma inferência simples no mundo de wumpus. Demonstra-se que $\neg P_{1,2}$ segue das quatro primeiras cláusulas da linha superior.

Sentença FNC → *Cláusula*$_1$ ∧· · · ∧ *Cláusula*$_n$
Cláusula → *Literal*$_1$ ∨· · · ∨ *Literal*$_m$
Literal → *Símbolo* | ¬*Símbolo*
Símbolo → *P* | *Q* | *R* | . . .
FormaCláusulaHorn → *FormaCláusulaDefinida* | *FormaCláusulaObjetivo*
FormaCláusulaDefinida → (*Símbolo*$_1$ ∧· · · ∧ *Símbolo*$_e$) ⇒ *Símbolo*
FormaCláusulaObjetivo → (*Símbolo*$_1$ ∧· · · ∧ *Símbolo*$_e$) ⇒ *Falso*

Figura 7.14 Uma gramática para a forma normal conjuntiva, cláusulas de Horn e cláusulas definidas. Uma cláusula tal como $A \wedge B$ => C ainda é uma cláusula definida quando é escrita como ¬ A ∨ ¬ B ∨ C, mas apenas a anterior é considerada a forma canônica para as cláusulas definidas. Uma outra classe é a sentença *k*-FNC, que é uma sentença de FNC, onde cada cláusula tem no máximo *k* literais.

Completude de resolução

Para concluir nossa discussão da resolução, mostraremos agora por que RESOLUÇÃO-LP é completa. Para isso, introduziremos o **fecho por resolução** $FR(S)$ de um conjunto de cláusulas S, que é o conjunto de todas as cláusulas deriváveis pela aplicação repetida da regra de resolução a cláusulas em S ou suas derivadas. O fecho por resolução é o que RESOLUÇÃO-LP calcula como

valor final da variável *cláusulas*. É fácil ver que *FR*(*S*) deve ser finito porque existe apenas um número finito de cláusulas distintas que podem ser construídas a partir dos símbolos $P_1, \ldots, P_k$ que aparecem em *S* (note que isso não seria verdadeiro sem a etapa de fatoração que remove várias cópias de literais). Consequentemente, RESOLUÇÃO-LP sempre termina.

O teorema da completude da resolução em lógica proposicional é chamado **teorema básico da resolução**:

Se um conjunto de cláusulas é não satisfatível, então o fecho por resolução dessas cláusulas contém a cláusula vazia.

Provamos esse teorema demonstrando sua contrapositiva: se o fecho *FR*(*S*) *não* contém a cláusula vazia, então *S* é satisfatível. De fato, podemos construir um modelo para *S* com valores-verdade adequados para $P_1,\ldots, P_k$. O procedimento de construção é:

Para *i* de 1 a *k*,

- Se existe uma cláusula em *FR*(*S*) contendo o literal $\neg P_i$ e todos os seus outros literais são falsos sob a atribuição escolhida para $P_1, \ldots, P_{i-1}$, então atribua *falso* a P_i.
- Caso contrário, atribua *verdadeiro* a P_i.

Essa atribuição para $P_1,\ldots, P_k$ é um modelo de *S*. Para verificar isso, assuma o oposto — que, em algum estágio *i* na sequência, atribuir o símbolo P_i faz com que alguma cláusula *C* torne-se falsa. Para que isso aconteça, todos os *outros* literais em *C* já devem ter sido falsificados por atribuições a P_1, $\ldots$, P_{i-1}. Assim, *C* deve agora parecer como ($falso \lor falso \lor \ldots falso \lor P_i$) ou como ($falso \lor falso \lor \ldots falso \lor \neg P_i$). Se houver apenas um desses dois em FR(*S*), o algoritmo irá atribuir o valor-verdade adequado para P_i tornar *C* verdadeiro, assim *C* só poderá ser falsificado se *ambas* as cláusulas estiverem em FR(*S*). Agora, uma vez que FR(*S*) é fechado sob resolução, vai conter o resolvente dessas duas cláusulas, e esse resolvente já terá todos os seus literais falsificados pelas atribuições a $P_1,\ldots, P_{i-1}$. Isso contradiz a nossa hipótese de que a primeira cláusula falsificada aparece no estágio *i*. Assim, provamos que a construção nunca falsifica uma cláusula em FR(*S*), ou seja, ela produz um modelo de FR(*S*) e, assim, um modelo de *S* em si (já que *S* está contido em FR(*S*)).

7.5.3 Cláusulas de Horn e cláusulas definidas

A completude da resolução a torna um método de inferência muito importante. Em muitas situações práticas, no entanto, o pleno poder de resolução não é necessário. Algumas bases de conhecimento do mundo real satisfazem certas restrições sobre a forma de sentenças que elas contêm, que permite que elas utilizem um algoritmo de inferência mais restrito e eficiente.

Uma destas formas restritas é a **cláusula definida**, que é uma disjunção de literais dos quais *exatamente um é positivo*. Por exemplo, a cláusula ($\neg L_{1,1} \lor \neg Brisa, \lor B_{1,1}$) é uma cláusula

definida, enquanto $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1})$ não é.

A **cláusula de Horn** é ligeiramente mais geral, uma disjunção de literais *dos quais pelo menos um é positivo*. Assim, todas as cláusulas definidas são cláusulas de Horn, como existem cláusulas sem literal positivo, que são chamadas cláusulas objetivo. As cláusulas de Horn são fechadas sob resolução: se você resolver duas cláusulas de Horn, receberá de volta uma cláusula de Horn.

As bases de conhecimento que contém apenas cláusulas definidas são interessantes por três razões:

1. Toda cláusula definida pode ser escrita como uma implicação cuja premissa é uma conjunção de literais positivos e cuja conclusão é um único literal positivo (veja o Exercício 7.13). Por exemplo, a cláusula definida $(\neg L_{1,1} \lor \neg Brisa \lor B_{1,1})$ pode ser escrita como implicação $(L_{1,1} \land Brisa) \Rightarrow B_{1,1}$. Na forma de implicação, a sentença é mais fácil de entender: informa que, se o agente está em [1,1] e há uma brisa, então [1,1] está com brisa. Na forma de Horn, a premissa é chamada **corpo**, e a conclusão, **cabeça**. Uma sentença que consiste em um único literal positivo, como $L_{1,1}$, é chamada de **fato**. Também pode ser escrita na forma de implicação como $Verdadeiro \Rightarrow L_{1,1}$, mas é mais simples escrever apenas $L_{1,1}$.
2. A inferência com cláusulas de Horn pode ser feita através de algoritmos de **encadeamento para a frente** e **encadeamento para trás**, que descreveremos a seguir. Ambos os algoritmos são naturais e, por isso, as etapas de inferência são óbvias e fáceis de os seres humanos seguirem. Esse tipo de inferência é a base para a **programação lógica**, que será discutida no Capítulo 9.
3. A decisão da consequência lógica com as cláusulas de Horn pode ser feita em tempo *linear* no tamanho da base do conhecimento — uma agradável surpresa.

7.5.4 Encadeamento para a frente e para trás

O algoritmo de encadeamento para a frente CONSEQUÊNCIA-LÓGICA-LP-EF?(BC, θ) determina se um único símbolo proposicional θ — a consulta — é consequência de uma base de conhecimento de cláusulas definidas. Ele começa a partir de fatos conhecidos (literais positivos) na base de conhecimento. Se todas as premissas de uma implicação forem conhecidas, sua conclusão será acrescentada ao conjunto de fatos conhecidos. Por exemplo, se $L_{1,1}$ e *Brisa* são conhecidas e $(L_{1,1} \land Brisa) \Rightarrow B_{1,1}$ está na base de conhecimento, então $B_{1,1}$ pode ser adicionada. Esse processo continua até a consulta θ ser acrescentada ou até não ser possível fazer inferências adicionais. O algoritmo detalhado é mostrado na Figura 7.15; o principal ponto a lembrar é que ele funciona em tempo linear.

função CONSEQUÊNCIA-LÓGICA-LP-EF? (BC, q) **retorna** *verdadeiro* ou *falso*
  entradas: BC, a base de conhecimento, um conjunto de cláusulas definidas proposicionais
    q, a consulta, um símbolo proposicional
  contagem ← uma tabela, onde contagem [c] é o número de símbolos nas premissas de c
  inferido ← uma tabela onde *inferido* [s] é *falso* inicialmente para todos os símbolos

```
  agenda ← uma lista de símbolos, inicialmente os símbolos reconhecidos como verdadeiros em BC

  enquanto agenda não é vazia faça
    p ← POP(agenda)
    se p = q então retornar verdadeiro
    se inferido [p] = falso então
      inferido[p] ← verdadeiro
      para cada cláusula c em BC onde p está em c.PREMISSA faça
        decrementar contagem[c]
        se contagem[c] = 0 então adicione c.CONCLUSÃO para agenda
  retornar falso
```

Figura 7.15 Algoritmo de encadeamento para a frente para lógica proposicional. A *agenda* controla os símbolos reconhecidos como verdadeiros, mas ainda "não processados". A tabela *contagem* controla a quantidade de premissas de cada implicação que ainda são desconhecidas. Sempre que um novo símbolo *p* da agenda for processado, a contagem será reduzida em uma unidade para cada implicação em cuja premissa *p* aparece (facilmente identificadas em tempo constante com indexação apropriada). Se a contagem chegar a zero, isso significa que todas as premissas da implicação são conhecidas e, assim, sua conclusão pode ser acrescentada à agenda. Finalmente, precisamos controlar quais símbolos foram processados; um símbolo que já está no conjunto de símbolos deduzidos não precisa ser adicionado à agenda novamente. Isso evita trabalho redundante e também previne repetições infinitas que poderiam ser causadas por implicações como $P \Rightarrow Q$ e $Q \Rightarrow P$.

O melhor caminho para entender o algoritmo é usar um exemplo e uma figura. A Figura 7.16(a) mostra uma base de conhecimento simples de cláusulas de Horn com *A* e *B* como fatos conhecidos. A Figura 7.16(b) mostra a mesma base de conhecimento desenhada como um **grafo E-OU** (veja o Capítulo 4). Em grafos E-OU, múltiplas arestas unidas por um arco indicam uma conjunção — toda aresta deve ser provada — enquanto múltiplas arestas sem arco indicam uma disjunção — qualquer aresta pode ser provada. É fácil ver como o encadeamento para a frente funciona no grafo. As folhas conhecidas (aqui, *A* e *B*) são definidas, e a inferência se propaga para cima no grafo, o mais longe possível. Onde quer que apareça uma conjunção, a propagação espera até todos os conjuntos serem conhecidos antes de prosseguir. O leitor deve examinar o exemplo em detalhes.

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

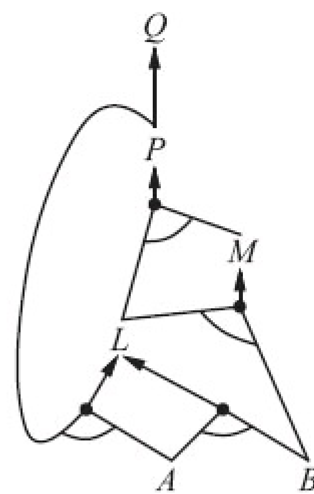
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

(a)



(b)

Figura 7.16 (a) Um conjunto de cláusulas de Horn. (b) Grafo E-OU correspondente.

É fácil ver que o encadeamento para a frente é **correto**: toda inferência é em essência uma aplicação de *Modus Ponens*. O encadeamento para a frente também é **completo**: toda sentença atômica permitida será derivada. O modo mais fácil de verificar isso é considerar o estado final da tabela *inferida* (depois que o algoritmo alcança um **ponto fixo** em que nenhuma nova inferência é possível). A tabela contém *verdadeiro* para cada símbolo inferido durante o processo e *falso* para todos os outros símbolos. Podemos visualizar a tabela como um modelo lógico; além disso, *toda cláusula definida na BC original é verdadeira nesse modelo*. Para verificar isso, suponha a afirmação oposta, ou seja, que alguma cláusula $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ seja falsa no modelo. Então, $a_1 \wedge \ldots \wedge a_k$ deve ser verdadeira no modelo e b dever ser falsa no modelo. Porém, isso contradiz nossa suposição de que o algoritmo alcançou um ponto fixo! Podemos concluir então que o conjunto de sentenças atômicas deduzidas no ponto fixo define um modelo da BC original. Mais ainda, qualquer sentença atômica θ que é consequência lógica da *BC* deve ser deduzida pelo algoritmo.

O encadeamento para a frente é um exemplo do conceito geral de raciocínio **orientado a dados** — isto é, o raciocínio em que o foco da atenção começa com os dados conhecidos. Ele pode ser usado em um agente para derivar conclusões a partir de percepções de entrada, frequentemente sem uma consulta específica em mente. Por exemplo, o agente de wumpus poderia informar (com TELL) suas percepções à base de conhecimento, utilizando um algoritmo de encadeamento para a frente incremental em que novos fatos pudessem ser adicionados à agenda para iniciar novas inferências. Em seres humanos, certa quantidade de raciocínio orientado a dados ocorre à medida que chegam novas informações. Por exemplo, se estou em um ambiente fechado e ouço a chuva começando a cair, pode me ocorrer que o piquenique será cancelado. Ainda assim, provavelmente não me ocorrerá que a décima sétima pétala da maior rosa no jardim do meu vizinho ficará molhada; os seres humanos mantêm o encadeamento para a frente sob cuidadoso controle, temendo ficar sobrecarregados com o acúmulo de consequências irrelevantes.

O algoritmo de encadeamento para trás, como seu nome sugere, funciona no sentido inverso a partir da consulta. Se a consulta θ é reconhecida como verdadeira, não é necessário nenhum trabalho. Caso contrário, o algoritmo encontra as implicações na base de conhecimento cuja conclusão é θ. Se for possível demonstrar que todas as premissas de uma dessas implicações são verdadeiras (por

encadeamento para trás), então θ é verdadeira. Quando aplicado à consulta Q da Figura 7.16, ele percorre o grafo no sentido inverso até alcançar um conjunto de fatos conhecidos que forme a base para uma prova. O algoritmo é em essência idêntico ao algoritmo de BUSCA-EM-GRAFOS-E-OU na Figura 4.11. Como ocorre no caso do encadeamento para a frente, uma implementação eficiente funciona em tempo linear.

O encadeamento para trás é uma forma de **raciocínio orientado por metas**. Ele é útil para responder a perguntas específicas como: "O que devo fazer agora?" e "Onde estão minhas chaves?". Com frequência, o custo do encadeamento para trás é *muito menor* que um custo linear em relação ao tamanho da base de conhecimento porque o processo só toca fatos relevantes.

7.6 VERIFICAÇÃO DE MODELOS PROPOSICIONAIS EFICIENTES

Nesta seção, descrevemos duas famílias de algoritmos eficientes para inferência proposicional geral, baseados na verificação de modelos: uma abordagem baseada na busca com retrocesso e outra na busca de subida de encosta local. Esses algoritmos fazem parte da "tecnologia" da lógica proposicional. Você poderá preferir folhear rapidamente esta seção no caso de uma primeira leitura do capítulo.

Os algoritmos que descrevemos se destinam à verificação da satisfatibilidade: o problema SAT. (Como anteriormente observado, o teste da consequência lógica $\alpha \models \beta$ pode ser realizado testando a não satisfatibilidade da $\alpha \land \neg\beta$.) Já notamos a conexão entre a localização de um modelo satisfatório para uma sentença lógica e a descoberta de uma solução para um problema de satisfação de restrições e, assim, talvez não seja surpresa o fato de as duas famílias de algoritmos se assemelharem bastante aos algoritmos de retrocesso da Seção 6.3 e aos algoritmos de busca local da Seção 6.4. Porém, eles são extremamente importante por si sós, porque muitos problemas combinatórios em ciência da computação podem ser reduzidos à verificação da satisfatibilidade de uma sentença proposicional. Qualquer melhoria em algoritmos de satisfatibilidade tem enormes consequências para nossa habilidade de tratar a complexidade em geral.

7.6.1 Um algoritmo com retrocesso completo

O primeiro algoritmo que examinaremos frequentemente é chamado **algoritmo de Davis-Putnam**, devido ao importante artigo de Martin Davis e Hilary Putnam (1960). De fato, o algoritmo é a versão descrita por Davis, Logemann e Loveland (1962), e, por essa razão, vamos chamá-lo DPLL, um acrônimo formado pelas iniciais dos quatro autores. O DPLL recebe como entrada uma sentença em forma normal conjuntiva — um conjunto de cláusulas. Como BUSCA-COM-RETROCESSO e CONSEQUÊNCIA-LÓGICA-TV?, ele é em essência uma enumeração recursiva em profundidade de modelos possíveis. Ele incorpora três melhorias em relação ao esquema simples de CONSEQUÊNCIA-LÓGICA-TV?:

- *Terminação prematura*: O algoritmo detecta se a sentença tem de ser verdadeira ou falsa,

mesmo no caso de um modelo parcialmente concluído. Uma cláusula é verdadeira se *qualquer* literal é verdadeiro, mesmo que os outros literais ainda não tenham valores-verdade; consequentemente, a sentença como um todo pode ser considerada verdadeira, mesmo antes de o modelo estar completo. Por exemplo, a sentença $(A \lor B) \land (A \lor C)$ é verdadeira se A é verdadeiro, independentemente dos valores de B e C. De modo semelhante, uma sentença é falsa se *qualquer* cláusula é falsa, o que ocorre quando cada um de seu literais é falso. Mais uma vez, isso pode ocorrer bem antes de o modelo estar completo. O término prematuro evita o exame de subárvores inteiras no espaço de busca.

- *Heurística de símbolo puro*: Um **símbolo puro** é um símbolo que sempre aparece com o mesmo "sinal" em todas as cláusulas. Por exemplo, nas três cláusulas $(A \lor \neg B)$, $(\neg B \lor \neg C)$ e $(C \lor A)$, o símbolo A é puro porque só aparece o literal positivo, B é puro porque só aparece o literal negativo e C é impuro. É fácil ver que, se uma sentença tem um modelo, ela tem um modelo com os símbolos puros atribuídos de forma a tornar seus literais *verdadeiros*, porque isso nunca poderá tornar uma cláusula falsa. Observe que, na determinação da pureza de um símbolo, o algoritmo pode ignorar cláusulas que já são reconhecidas como verdadeiras no modelo construído até o momento. Por exemplo, se o modelo contém $B = falso$, a cláusula $(\neg B \lor \neg C)$ já é verdadeira, e nas cláusulas C restantes aparece apenas um literal positivo; por essa razão, C torna-se puro.
- *Heurística de cláusula unitária*: Uma **cláusula unitária** foi definida anteriormente como uma cláusula com apenas um literal. No contexto de DPLL, essa expressão também significa cláusulas em que todos os literais com exceção de um já têm o valor *falso* atribuído pelo modelo. Por exemplo, se o modelo contém $B = verdadeiro$, então $(\neg B \lor \neg C)$ simplifica para $\neg C$, que é uma cláusula unitária. É óbvio que, para que essa cláusula seja verdadeira, C deve ser definido como *falso*. A heurística de cláusula unitária atribui todos esses símbolos antes de efetuar a ramificação sobre o restante. Uma consequência importante da heurística é que qualquer tentativa de provar (por refutação) um literal que já está na base de conhecimento terá sucesso imediato (Exercício 7.23). Note também que a atribuição de uma cláusula unitária pode criar outra cláusula unitária — por exemplo, quando C é definido como *falso*, $(C \lor A)$ se torna uma cláusula unitária, fazendo com que o valor verdadeiro seja atribuído a A. Essa "cascata" de atribuições forçadas é chamada **propagação unitária**. Ela lembra o processo de encadeamento para a frente com cláusulas definidas e, na realidade, se a expressão em FNC contém apenas cláusulas definidas, DPLL essencialmente reproduz o encadeamento para a frente (veja o Exercício 7.24).

O algoritmo DPLL é mostrado na Figura 7.17, o que oferece o esqueleto essencial do processo de busca.

O que a Figura 7.17 não mostra são os truques que tornam os resolvedores SAT escaláveis para problemas maiores.

função SATISFATÍVEL-DPLL?(s) **retorna** *verdadeiro* ou *falso*
  entradas: *s*, uma sentença em lógica proposicional

  cláusulas ← o conjunto de cláusulas na representação em FNC de *s*

```
    símbolos ← uma lista dos símbolos proposicionais em s
    retornar DPLL(cláusulas, símbolos, { })

função DPLL(cláusulas, símbolos, modelo) retorna verdadeiro ou falso

    se toda cláusula em cláusulas é verdadeira em modelo então retornar verdadeiro
    se alguma cláusula em cláusulas é falsa em modelo então retornar falso
    P, valor ← ENCONTRAR-SÍMBOLO-PURO(símbolos, cláusulas, modelo)
    se P é não nulo então retornar DPLL(cláusulas, símbolos – P, modelo ∪ {P = valor})
    P, valor ← ENCONTRAR-CLÁUSULA-UNITÁRIA(cláusulas, modelo)
    se P é não nulo então retornar DPLL(cláusulas, símbolos – P, modelo ∪ {P = valor})
    P ← PRIMEIRO(símbolos); restantes ← RESTO(símbolos)
    retornar DPLL(cláusulas, restantes, modelo ∪ {P = verdadeiro}) ou
                                   DPLL(cláusulas, restantes, modelo ∪ {P = falso})
```

Figura 7.17 O algoritmo DPLL para verificar a satisfatibilidade de uma sentença em lógica proposicional. As ideias contidas em ENCONTRAR-SÍMBOLO-PURO e ENCONTRAR-CLÁUSULA-UNITÁRIA são descritas no texto; cada um deles retorna um símbolo (ou nulo) e o valor-verdade que deve ser atribuído a esse símbolo. Como CONSEQUÊNCIA-LÓGICA-TV?, DPLL opera sobre modelos parciais.

É interessante que a maioria desses truques na verdade são bastante gerais, e já os vimos em outras formas:

1. **Análise de componentes** (como visto na Tasmânia em PSRs (ver Capítulo 6)): Como o DPLL atribui valores-verdade para variáveis, o conjunto de cláusulas pode ficar separado em subconjuntos disjuntos, chamados de **componentes,** que não compartilham as variáveis não atribuídas. Havendo uma maneira eficiente para detectar quando isso ocorre, um solucionador pode ganhar uma velocidade considerável, trabalhando em cada componente separadamente.
2. **Variável e ordenação de valor** (como visto na Seção 6.3.1 para PSRs): A nossa implementação simples de DPLL utiliza uma ordenação de variáveis arbitrárias e sempre tenta o valor verdadeiro antes do *falso*. A **heurística de grau** sugere escolher a variável que aparece com mais frequência sobre todas as demais cláusulas.
3. **Retrocesso inteligente** (como visto na Seção 6.3 para PSRs): Muitos problemas que não podem ser resolvidos em horas de tempo de execução com retrocesso cronológico podem ser resolvidos em segundos com retrocesso inteligente que retrocede todo o caminho até o ponto relevante de conflito. Todos os resolvedores que realizam retrocesso inteligente usam alguma forma de **aprendizagem de cláusula de conflito** para gravar os conflitos de modo que não se repitam mais tarde na busca. Geralmente um conjunto limitado de tamanho de conflitos é mantido, e raramente os descartados são utilizados.
4. **Reinícios aleatórios**: Às vezes parece que uma execução não apresenta progresso. Nesse caso, podemos começar de novo a partir do topo da árvore de busca, em vez de tentar continuar. Depois do reinício são feitas escolhas aleatórias diferentes (na variável e na seleção do valor).

As cláusulas que são aprendidas no decorrer da primeira execução são mantidas após o reinício e podem ajudar a podar o espaço de busca. O reinício não garante que uma solução será encontrada mais rápido, mas reduz a variância em tempo para a solução.

5. **Indexação inteligente** (como visto em muitos algoritmos): Os métodos em si utilizados para aceleração em DPLL, bem como os truques utilizados pelos solucionadores modernos, exigem a rápida indexação de coisas como "o conjunto de cláusulas na qual a variável X_i aparece como um literal positivo". Essa tarefa é complicada pelo fato de que os algoritmos estão interessados apenas nas cláusulas que ainda não foram satisfeitas por atribuições a variáveis anteriores, então as estruturas de indexação devem ser atualizadas dinamicamente à medida que a computação prossegue.

Com esses avanços, os solucionadores modernos podem lidar com problemas de dezenas de milhões de variáveis. Eles revolucionaram áreas como a de verificação de hardware e de protocolo de segurança, que antes exigiam provas trabalhosas, feitas à mão.

7.6.2 Algoritmos de busca local

Até agora vimos vários algoritmos de busca local neste livro, incluindo SUBIDA-DE-ENCOSTA e TÊMPERA-SIMULADA. Esses algoritmos podem ser aplicados diretamente a problemas de satisfatibilidade, desde que seja escolhida a função de avaliação correta. Como o objetivo é encontrar uma atribuição que satisfaça a toda cláusula, uma função de avaliação que efetue a contagem do número de cláusulas não satisfeitas fará o trabalho. De fato, essa é exatamente a medida usada pelo algoritmo CONFLITOS-MÍNIMOS para PSRs. Todos esses algoritmos executam etapas no espaço de atribuições completas, invertendo o valor-verdade de um símbolo de cada vez. Normalmente, o espaço contém muitos mínimos locais e são exigidas várias formas de aleatoriedade para escapar desses mínimos locais. Nos últimos anos, houve um grande volume de experimentos com a finalidade de descobrir um bom equilíbrio entre o caráter ambicioso e a aleatoriedade.

Um dos mais simples e mais eficientes algoritmos a emergir de todo esse trabalho é chamado WALKSAT (Figura 7.18). Em toda iteração, o algoritmo seleciona uma cláusula não satisfeita e um símbolo na cláusula a ser invertido. Ele escolhe ao acaso entre dois modos de selecionar o símbolo a ser invertido: (1) uma etapa de "conflitos mínimos", que minimiza o número de cláusulas não satisfeitas no novo estado, e (2) uma etapa de "percurso aleatório", que seleciona o símbolo aleatoriamente.

função WALKSAT(*cláusulas*, *p*, *inversões_max*) **retorna** um modelo satisfatório ou *falha*
  entradas: *cláusulas*, um conjunto de cláusulas em lógica proposicional
    p, a probabilidade de optar por realizar um movimento de "percurso aleatório", normalmente em torno de 0,5
    inversões_max, número de inversões permitidas antes de desistir

  modelo ← uma atribuição aleatória de *verdadeiro*/*falso* aos símbolos de *cláusulas*

para i = 1 **até** *inversões_max* **faça**
se *modelo* satisfaz *cláusulas* **então retornar** *modelo*
cláusula ← uma cláusula selecionada ao acaso de *cláusulas* que é falsa em *modelo*
com probabilidade p, inverter o valor de *modelo* de um símbolo selecionado ao acaso de *cláusula*
senão inverter qualquer símbolo em *cláusula* que maximize o número de cláusulas satisfeitas
retornar *falha*

Figura 7.18 Algoritmo WALKSAT para verificar a satisfatibilidade pela inversão aleatória dos valores de variáveis. Existem muitas versões do algoritmo.

Quando o WALKSAT retorna um modelo, a sentença de entrada é realmente satisfatível, mas quando retorna *falha* existem duas causas possíveis: ou a sentença é insatisfatível ou precisamos dar mais tempo ao algoritmo. Se inicializarmos *inversões_max* = ∞ e $p > 0$, o WALKSAT vai retornar eventualmente um modelo (se existir) porque as etapas de percurso aleatório acabarão por atingir a solução. Infelizmente, se *inversões_max* for infinito e se a sentença for não satisfatível, o algoritmo nunca terminará!

Por essa razão, o WALKSAT é mais útil quando esperamos que exista uma solução — por exemplo, os problemas discutidos nos Capítulos 3 e 6 em geral têm soluções. Por outro lado, o WALKSAT nem sempre pode detectar a *não satisfatibilidade*, necessária para definir a consequência lógica. Por exemplo, um agente não pode utilizar o WALKSAT para provar que um quadrado é seguro no mundo de wumpus. Em vez disso, ele pode dizer: "Pensei durante uma hora e não consegui descobrir um mundo possível em que o quadrado *não fosse* seguro." Esse pode ser um bom indicador empírico de que o quadrado é seguro, mas certamente não é uma prova.

7.6.3 O cenário dos problemas SAT aleatórios

Alguns problemas são mais difíceis que outros. Problemas *fáceis* podem ser resolvidos por qualquer algoritmo antigo, mas, por sabermos que SAT é NP-completo, pelo menos algumas instâncias de problema exigem tempo de execução exponencial. No Capítulo 6, vimos algumas descobertas surpreendentes sobre alguns tipos de problemas. Por exemplo, o problema de n-rainhas, que se pensava ser bastante complicado para os algoritmos de busca por retrocesso, acabou por ser trivialmente simples para os métodos de busca local, como de conflitos mínimos. Isso deveu-se às soluções estarem muito densamente distribuídas no espaço das atribuições e à garantia de que qualquer atribuição inicial teria uma solução nas proximidades. Assim, n-rainhas é fácil por ser **sub-restrito**.

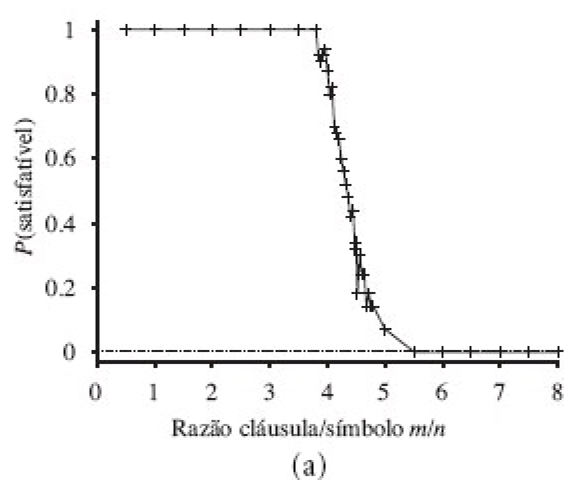
Quando encaramos problemas de satisfatibilidade na forma normal conjuntiva, um problema sub-restrito é aquele que tem relativamente *poucas* cláusulas restringindo as variáveis. Por exemplo, observe esta sentença 3-FNC gerada aleatoriamente com cinco símbolos e cinco cláusulas:

$$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E)$$
$$\wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C).$$

Dezesseis das 32 atribuições possíveis são modelos dessa sentença; assim, em média, seriam necessárias apenas duas conjeturas aleatórias para encontrar um modelo. Esse é um problema fácil de satisfatibilidade, como é a maioria dos problemas sub-restritos. Por outro lado, um problema *super-restrito* tem muitas cláusulas relativas ao número de variáveis e é provável que não tenha soluções.

Para ir além dessas intuições básicas, é preciso definir exatamente como as sentenças aleatórias são geradas. A notação $FNC_k\ (m, n)$ indica uma sentença k-FNC com m cláusulas e n símbolos, onde as cláusulas são escolhidas de maneira uniforme, independentemente, e sem substituição dentre todas as cláusulas com literais k diferentes, que são positivos ou negativos aleatoriamente. (Um símbolo não pode aparecer duas vezes em uma cláusula nem uma cláusula pode aparecer duas vezes em uma sentença.)

Dada uma fonte de sentenças aleatórias, podemos medir a probabilidade de satisfatibilidade. A Figura 7.19(a) demarca a probabilidade de $FNC_3\ (m, 50)$, isto é, sentenças com 50 variáveis e três literais por cláusula, como uma função da razão de cláusula/símbolo, m/n. Como esperado, com m/n baixo, a probabilidade de satisfatibilidade é próximo de 1, e, para um m/n alto, a probabilidade fica próxima de 0. A probabilidade cai acentuadamente em torno de $m/n = 4{,}3$. Achamos empiricamente que o "precipício" fica aproximadamente no mesmo lugar (para $k = 3$) e fica cada vez mais pronunciado à medida que n aumenta. Teoricamente, a **suposição do limiar de satisfatibilidade** afirma que, para cada $k \geq 3$, há uma razão limite de r_k de tal forma que, quando n tende ao infinito, a probabilidade de que $FNC_k\ (n, rn)$ seja satisfatível se torna 1 para todos os valores de r abaixo do limite e 0 para todos os valores acima. A suposição ainda não foi provada.



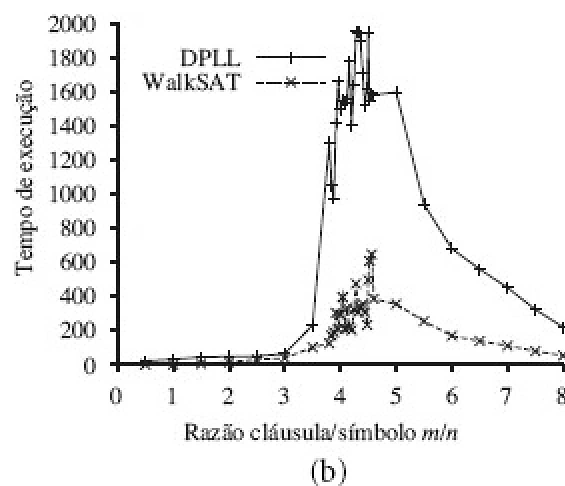


Figura 7.19 (a) Gráfico mostrando a probabilidade de uma sentença 3-FNC aleatória com $n = 50$ símbolos ser satisfatível, como uma função da razão cláusula/símbolo m/n. (b) Gráfico do tempo mediano de execução (medido em número de chamadas recursivas da DPLL, um bom parâmetro *proxy*) de 3-FNC sentenças aleatórias. Os problemas mais difíceis têm uma razão cláusula/símbolo de cerca de 4.3.

Agora que temos uma boa ideia de onde estão os problemas satisfatíveis e insatisfatíveis, a pergunta é a seguinte: onde estão os problemas difíceis? Acontece que eles também estão muitas vezes no valor limite. A Figura 7.19(b) mostra que 50 problemas de símbolo no valor limiar de 4,3 são cerca de 20 vezes mais difíceis de resolver do que aqueles com proporção de 3,3. Os problemas

sub-restritos são os mais fáceis de resolver (porque é muito fácil adivinhar uma solução), os problemas super-restritos não são tão fáceis como os sub-restritos, mas ainda são muito mais fáceis do que os que estão no limiar.

7.7 AGENTES BASEADOS EM LÓGICA PROPOSICIONAL

Nesta seção, vamos reunir o que aprendemos até agora, a fim de construir agentes do mundo de wumpus que utilizam lógica proposicional. O primeiro passo é permitir que o agente deduza, na medida do possível, o estado do mundo, dada a sua história de percepção. Isso requer escrever um modelo lógico completo dos efeitos das ações. Podemos mostrar também como o agente pode acompanhar o mundo de forma eficiente sem voltar na história da percepção para cada inferência. Finalmente, mostraremos como o agente pode utilizar a inferência lógica para a construção de planos garantidos de atingir os seus objetivos.

7.7.1 O estado atual do mundo

Como dito no início do capítulo, um agente lógico opera deduzindo o que fazer a partir de uma base de conhecimento de sentenças sobre o mundo. A base de conhecimento é composta de axiomas — conhecimento geral sobre como o mundo funciona — e sentenças de percepção obtidas da experiência do agente em um mundo particular. Nesta seção, vamos nos concentrar no problema de deduzir o estado atual do mundo de wumpus — onde estou, esse quadrado é seguro, e assim por diante.

Na Seção 7.4.3 começamos a coletar axiomas. O agente sabe que o quadrado inicial não contém poço ($\neg P_{1,1}$) e nenhum wumpus ($\neg W_{1,1}$). Além disso, para cada quadrado, ele sabe que o quadrado está com vento se e somente se um quadrado vizinho tiver poço; e um quadrado terá mau cheiro se e somente se um quadrado vizinho tiver um wumpus. Assim, incluiremos uma grande coleção de sentenças da seguinte forma:

$$
\begin{aligned}
&B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}) \\
&S_{1,1} \Leftrightarrow (W_{1,2} \vee W_{2,1}) \\
&\ldots
\end{aligned}
$$

O agente também sabe que existe exatamente um wumpus. Isso está expresso em duas partes. Primeiro, precisamos informar que há pelo menos um wumpus:

$$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,3} \vee W_{4,4}.$$

Então, informamos que há, *no máximo*, um wumpus. Para cada par de locais, adicionamos uma sentença informando que pelo menos uma delas deve estar livre do wumpus:

$$\neg W_{1,1} \vee \neg W_{1,2}$$
$$\neg W_{1,1} \vee \neg W_{1,3}$$
$$\dots$$
$$\neg W4{,}3 \vee \neg W_{4,4}.$$

Até aí, tudo bem. Agora vamos considerar as percepções do agente. Se houver atualmente mau cheiro, pode-se supor que uma proposição *mau cheiro* deverá ser adicionada à base de conhecimento. Entretanto, isso não é tão certo: se não havia mau cheiro na etapa de tempo anterior, então ¬*Mau cheiro* já teria sido reconhecido, e a nova afirmação resultaria simplesmente em uma contradição. O problema será resolvido quando percebermos que uma percepção afirma algo *somente sobre o tempo atual*. Assim, se a etapa do tempo (como provido em CRIAR-SENTENÇA-DE-PERCEPÇAO na Figura 7.1) for 4, então somamos o $Mau\ cheiro^4$ à base de conhecimento, em vez do *Mau cheiro* — evitando nitidamente qualquer contradição com $\neg Mau\ cheiro^3$. O mesmo vale para as percepções de brisa, impacto, brilho e gritos.

A ideia de associar proposições com etapas de tempo se estende a qualquer aspecto do mundo que muda ao longo do tempo. Por exemplo, a base de conhecimento inicial inclui $L^0_{1,1}$ — o agente está no quadrado [1,1] no tempo 0 —, bem como $OlhandoParaLeste^0$, $TemSeta^0$ e $WumpusVivo^0$. Utilizamos a palavra **fluente** (do latim *fluens*, fluindo) para nos referir a um aspecto do mundo que muda. “Fluente” é sinônimo de “variável de estado”, no sentido descrito na discussão das representações fatoradas na Seção 2.4.7. Os símbolos associados aos aspectos permanentes do mundo não necessitam de um sobrescrito de tempo e às vezes são chamados de **variáveis atemporais**.

Podemos conectar as percepções de mau cheiro e de brisa diretamente às propriedades dos quadrados onde são experimentados através da localização fluente como segue.[10] Para qualquer etapa t e qualquer quadrado [*x, y*], afirmamos

$$L^t_{x,y} \Rightarrow (Brisa^t \Leftrightarrow B_{x,y})$$
$$L^t_{x,y} \Rightarrow (Mau\ cheiro^t \Leftrightarrow S_{x,y}).$$

Agora, é claro, precisamos de axiomas que permitam que o agente mantenha o controle de fluentes como $L^t_{x,y}$. Esses fluentes se alteram como resultado de ações tomadas pelo agente; então, na terminologia do Capítulo 3, é preciso anotar o **modelo de transição** do mundo de wumpus como um conjunto de sentenças lógicas.

Primeiro, precisamos de símbolos proposicionais para as ocorrências de ações. Tal como acontece com as percepções, esses símbolos são indexados pelo tempo, portanto, $ParaFrente^0$ significa que o agente executa a ação *ParaFrente* no instante 0. Por convenção, a percepção para um determinado instante acontece primeiro, seguida pela ação para aquele instante, seguida de uma transição para o próximo instante.

Para descrever como o mundo muda, podemos tentar escrever **axiomas de efeito** que especificam o resultado de uma ação no instante seguinte. Por exemplo, se o agente estiver em um local [1,1] voltado para o leste no instante 0 e se move *ParaFrente*, o resultado é que o agente estará no quadrado [2,1] e não mais em [1,1]:

$$L^0_{1,1} \wedge EncarandoLeste^0 \wedge Para\ a\ frente^0 \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,1}). \tag{7.1}$$

Precisaríamos de uma sentença dessa para cada instante possível, para cada um dos 16 quadrados e cada uma das quatro orientações. Precisaríamos também de sentenças semelhantes para outras ações: *Agarrar*, *Atirar*, *Escalar*, *VirarEsquerda* e *VirarDireita*.

Vamos supor que o agente decida se mover (*ParaFrente*) no tempo 0 e declare esse fato em sua base de conhecimento. Dado o axioma de efeito na Equação 7.1, combinado com as afirmações iniciais sobre o estado no tempo 0, o agente já pode deduzir que está em [2,1]. Ou seja, ASK (*BC*, $L^1_{2,1}$) = *verdadeiro*. Até agora, tudo bem. Infelizmente, a notícia em outros lugares não é tão boa: se ASK (perguntarmos) (*BC*, $TemSeta^1$), a resposta será *falso,* isto é, o agente não pode provar que ainda tem a seta ou que não a tem! A informação foi perdida devido ao axioma de efeito falhar em declarar o que permaneceu *inalterado* como resultado de uma ação. A necessidade de fazer isso dá origem ao **problema de persistência**.[11] Uma possível solução para o problema de persistência seria adicionar **axiomas de persistência** afirmando explicitamente todas as proposições que permanecem as mesmas. Por exemplo, para cada tempo *t*, teríamos

$$ParaFrente^t \Rightarrow (TerFlecha^t \Leftrightarrow TerFlecha^{t+1})$$
$$ParaFrente^t \Rightarrow (WumpusVivo^t \Leftrightarrow WumpusVivo^{t+1})$$

onde mencionamos explicitamente cada proposição que permanece inalterada do tempo *t* para o tempo *t* + 1, sob a ação *ParaFrente*. Embora o agente agora saiba que ainda tem a flecha depois de mover para a frente e que o wumpus não morreu ou voltou à vida, a proliferação de axiomas de persistência parece notavelmente ineficiente. Em um mundo com *m* ações diferentes e *n* fluentes, o conjunto de axiomas de persistência será do tamanho *O*(*mn*). Essa manifestação específica do problema de persistência algumas vezes é chamada de **problema de persistência representacional**. Historicamente, o problema foi um dos mais importantes para os pesquisadores de IA, e ainda será explorado nas notas no final do capítulo.

O problema de persistência representacional é importante porque o mundo real tem muitos fluentes, para dizer o mínimo. Felizmente para nós, seres humanos, tipicamente cada ação não muda mais do que um pequeno número *k* desses fluentes — o mundo apresenta **localidade**. A solução do problema de persistência representacional exige a definição do modelo de transição, com um conjunto de axiomas de tamanho *O*(*mk*) em vez do tamanho *O*(*mn*). Há também um **problema de persistência inferencial**: o problema de projetar para a frente os resultados de um plano de ação da etapa *t* em tempo *O*(*kt*), em vez de *O*(*nt*).

A solução para o problema envolve uma mudança de foco de escrever axiomas sobre as *ações* para escrever axiomas sobre *fluentes.* Assim, para cada fluente *F*, teremos um axioma que define o valor verdade de F^{t+1} em termos de fluentes (incluindo o próprio *F*) no instante *t* e as ações que possam ter ocorrido no instante *t*. Agora, o valor-verdade de F^{t+1} pode ser definido em uma das duas formas: ou a ação no instante *t* faz com que *F* seja verdade em *t* + 1 ou *F* já era verdade no instante *t* e a ação no instante *t* não faz com que seja falsa. Um axioma dessa forma é chamado de **axioma de estado sucessor** e tem este esquema:

$$F^{t+1} \Leftrightarrow AçãoCausaF^t \lor (F^t \land \neg AçãoCausaNãoF^t).$$

Um dos axiomas mais simples de estado sucessor é o *TemFlecha.* Por não haver nenhuma ação para recarregar, a parte $AçãoCausaF^t$ vai embora e ficamos com

$$TemFlecha^{t+1} \Leftrightarrow (TemFlecha^t \wedge \neg Atira^t). \quad (7.2)$$

Para a localização do agente, os axiomas de estado sucessor são mais elaborados. Por exemplo, $L_{1,1}^{t+1}$ é verdade se (a) o agente se moveu *ParaFrente* de [1,2], quando voltado para o sul, ou de [2,1], quando voltado para o oeste; ou (b) $L^t_{1,1}$ já era verdadeiro e a ação não causou movimento (ou porque a ação não era *ParaFrente* ou porque esbarrou em um muro). Escrito em lógica proposicional, torna-se

$$\begin{aligned} L_{1,1}^{t+1} \quad &\Leftrightarrow \quad (L_{1,1}^t \wedge (\neg ParaFrente^t \vee Impacto^{t+1})) \\ &\Leftrightarrow \quad (L_{1,2}^t \wedge (Sul^t \wedge ParaFrente^t)) \\ &\Leftrightarrow \quad (L_{2,1}^t \wedge (Oeste^t \wedge ParaFrente^t)). \end{aligned} \quad (7.3)$$

O Exercício 7.26 pede para você escrever axiomas para os fluentes restantes do mundo wumpus.

Dado um conjunto completo de axiomas de estado sucessor e os outros axiomas listados no início desta seção, o agente será capaz de PERGUNTAR (ASK) e responder qualquer pergunta que tenha uma resposta sobre o estado atual do mundo. Por exemplo, na Seção 7.2 a sequência inicial de percepções e ações é

$$\begin{array}{l} \neg Mau\ cheiro^0 \wedge \neg Brisa^0 \wedge \neg Brilho^0 \wedge \neg Impacto^0 \wedge \neg Grito^0 \ ; \ ParaFrente^0 \\ \neg Mau\ cheiro^1 \wedge Brisa^1 \wedge \neg Brilho^1 \wedge \neg Impacto^1 \wedge \neg Grito^1 \ ; \ VirarDireita^1 \\ \neg Mau\ cheiro^2 \wedge Brisa^2 \wedge \neg Brilho^2 \wedge \neg Impacto^2 \wedge \neg Grito^2 \ ; \ VirarDireita^2 \\ \neg Mau\ cheiro^3 \wedge Brisa^3 \wedge \neg Brilho^3 \wedge \neg Impacto^3 \wedge \neg Grito^3 \ ; \ ParaFrente^3 \\ \neg Mau\ cheiro^4 \wedge \neg Brisa^4 \wedge \neg Brilho^4 \wedge \neg Impacto^4 \wedge \neg Grito^4 \ ; \ VirarDireita^4 \\ \neg Mau\ cheiro^5 \wedge \neg Brisa^5 \wedge \neg Brilho^5 \wedge \neg Impacto^5 \wedge \neg Grito^5 \ ; \ ParaFrente^5 \\ Mau\ cheiro^6 \wedge \neg Brisa^6 \wedge \neg Brilho^6 \wedge \neg Impacto^6 \wedge \neg Grito^6 \end{array}$$

Neste ponto, temos ASK (BC, $L_{2,2}^6$) = *verdadeiro*, de modo que o agente sabe onde está. Além disso, ASK (BC, $W_{1,3}$) = *verdadeiro* e ASK (BC, $P_{3,1}$) = *verdadeiro*, assim o agente encontrou o wumpus e um dos poços. A questão mais importante para o agente é se um quadrado está OK para entrar, ou seja, o quadrado não contém poço nem wumpus vivo. É conveniente acrescentar os axiomas para isso, tendo a forma de

$$OK_{x,y}^t \quad \Leftrightarrow \quad \neg P_{x,y} \wedge \neg(W_{x,y} \wedge WumpusVivo^t).$$

Finalmente, ASK (BC, $OK_{2,2}^6$) = *verdadeiro*, de modo que o quadrado [2, 2] está OK para entrar. De fato, dado um algoritmo de inferência consistente e completo, como o DPLL, o agente pode responder a qualquer questão que seja respondível sobre quais quadrados estão OK — e pode fazê-lo em apenas alguns milissegundos para mundos de wumpus pequenos a médios.

Resolver os problemas representacionais e inferenciais de persistência é um grande passo à frente, mas um problema pernicioso permanece: é preciso confirmar que *todas as* precondições necessárias de uma ação empregada sejam satisfeitas para que ela tenha o efeito pretendido. Dissemos que a ação *ParaFrente* move o agente para a frente, a menos que haja um muro no caminho, mas existem muitas outras exceções incomuns que podem causar a falha da ação: o agente pode tropeçar e cair, ser acometido de um ataque cardíaco, ser carregado por morcegos gigantes etc. O **problema de qualificação** é a especificação de todas essas exceções. Não há solução completa dentro da lógica; os projetistas de sistema têm que usar o bom-senso para decidir o quanto de detalhe desejam na

especificação de seu modelo e que detalhes querem deixar de fora. Veremos no Capítulo 13 que a teoria da probabilidade nos permite resumir todas as exceções sem nomeá-las explicitamente.

7.7.2 Um agente híbrido

A capacidade de deduzir vários aspectos do estado do mundo pode ser combinada de forma bastante simples com regras de condição-ação e com os algoritmos de resolução de problemas dos Capítulos 3 e 4 para produzir um **agente híbrido** para o mundo wumpus. A Figura 7.20 mostra uma forma possível de fazer isso. O programa do agente mantém e atualiza uma base de conhecimento, bem como um plano atual. A base de conhecimento inicial contém os axiomas *atemporais*, os que não dependem de t, como o axioma relacionando à falta de brisa dos quadrados com a presença de poços. Em cada instante, a nova sentença percebida é adicionada juntamente com todos os axiomas que dependem de t, como os axiomas de estado sucessor (a próxima seção explica por que o agente não precisa de axiomas para instantes futuros). Então, o agente utiliza inferência lógica, perguntando (ASKing) sobre a base de conhecimento, para planejar quais quadrados são seguros e quais ainda têm que ser visitados.

```
função AGENTE-WUMPUS-HIBRIDO(percepção) retorna uma ação
  entradas: percepção, uma lista, [fedor,brisa,brilho,impacto,grito]
  persistente: BC, uma base de conhecimento, inicialmente o “wumpus físico” atemporal t, um con
                 indicando o plano de tempo, uma sequência de ação, inicialmente vazia

  TELL(BC,CRIAR-SENTENÇA-DE-PERCEPÇÃO(percepção, t))
  TELL a BC as sentenças temporais de “física” para o instante t
  salvar ←{[x, y] : ASK(BC, OK^t_{x,y}) = verdadeiro}
  se ASK(BC, Brilho^t) = verdadeiro então
    plano ←[Agarrar] + PLANO-ROTA(atual, {[1,1]}, seguro) + [Escalar]
  se plano é vazio então
    não visitado ←{[x, y] : ASK(BC,L^{t'}_{x,y}) = falso para todo t'≤ t}
    plano ← PLANO-ROTA(atual, não visitado □ seguro, seguro)
  se plano é vazio e ASK(BC,TemFlecha^t) = verdadeiro então
    wumpus-possível ←{[x, y] : ASK(BC,¬W_{x,y}) = falso}
    plano ←PLANO-ATIRAR(atual, wumpus_possível, seguro)
  se plano é vazio então // única chance arriscar
    não_arriscado ←{[x, y] : ASK(BC,¬OK^t_{x,y}) = falso}
    plano ← PLANO-ROTA(atual, não visitado □ não_seguro, seguro)
  se plano é vazio então
    plano ← PLANO-ROTA(atual, {[1, 1]}, seguro) + [Escalar]
  ação ← POP(plano)
  TELL(BC,CRIAR-SENTENÇA-AÇÃO(ação, t))
  t ← t + 1
```

```
    retornar ação

função PLANO-ROTA(atual,objetivo,permitido) retornar uma sequência de ação
    entradas: atual, posição atual do agente
                objetivos, um conjunto de quadrados; tentar planejar uma rota para um deles
                permitido, um conjunto de quadrados que podem formar parte da rota

    problema ← PROBLEMA-ROTA(atual,objetivos,permitido)
    retornar BUSCA-A*-EM-GRAFOS(problema)
```

Figura 7.20 Um programa de agente híbrido para o mundo de wumpus. Ele utiliza uma base de conhecimento proposicional para inferir o estado do mundo e uma combinação de busca de resolução de problemas e código de domínio específico para decidir que ações tomar.

O corpo principal do programa do agente constrói um plano baseado em uma prioridade decrescente de objetivos. Primeiro, se houver brilho, o programa constrói um plano para agarrar o ouro, seguir uma rota de volta ao local inicial e sair da caverna. Caso contrário, se não houver nenhum plano atual, o programa planeja uma rota para o próximo quadrado seguro que ainda não visitou, certificando-se de que a rota passa apenas por quadrados seguros. O planejamento de rotas é feito com uma busca A*, não com ASK. Se não houver quadrados seguros para explorar, a próxima etapa — se o agente ainda tiver uma flecha — é tentar deixar um quadrado seguro atirando em um dos possíveis locais de wumpus. Isso será determinado perguntando onde ASK (BC, $\neg W_{x, y}$) é falso, ou seja, onde *não* se sabe que *não* existe um wumpus. A função PLANO-TIRO (não mostrada) utiliza PLANO-ROTA para planejar uma sequência de ações que vai alinhar esse tiro. Se falhar, o programa vai procurar um quadrado para explorar que não seja comprovadamente inseguro, isto é, um quadrado para o qual ASK (BC, $\neg OK^t_{x,y}$) retorne falso. Se não houver tal quadrado, então a missão é impossível, e o agente retira-se para [1,1] e sai da caverna.

7.7.3 Estimação de estado lógico

O programa do agente na Figura 7.20 funciona muito bem, mas tem uma grande fragilidade: com o passar do tempo, o custo computacional envolvido nas chamadas ASK aumenta cada vez mais. Isso acontece principalmente porque as inferências necessárias têm de voltar mais e mais no tempo e envolvem mais e mais símbolos proposicionais. Obviamente, isso é insustentável — não podemos ter um agente cujo tempo para processar cada percepção cresce em proporção ao comprimento da sua vida! O que realmente precisamos é de uma atualização de tempo *constante*, isto é, independente de t. A resposta óbvia é salvar ou colocar em **cache** os resultados de inferência, de modo que o processo de inferência no próximo instante possa ser construído sobre os resultados das etapas anteriores, em vez de ter de recomeçar novamente do zero.

Como vimos na Seção 4.4, a história passada de percepções e todas as suas ramificações pode ser substituída pelo **estado de crença**, isto é, alguma representação do conjunto de todos os estados

atuais possíveis do mundo.[12] O processo de atualização do estado de crença como novas percepções é chamado de **estimação de estado**. Considerando que, na Seção 4.4, o estado de crença era uma lista explícita de estados, aqui podemos utilizar uma sentença lógica envolvendo os símbolos de proposição associados com a etapa de tempo atual, bem como os símbolos atemporais. Por exemplo, a sentença lógica

$$WumpusVivo^1 \wedge L^1_{2,1} \wedge B_{2,1} \wedge (P_{3,1} \vee P_{2,2}) \tag{7.4}$$

representa o conjunto de todos os estados no tempo 1 em que o wumpus está vivo, o agente está em [2,1], esse quadrado tem brisa e há um poço em [3,1] ou [2, 2], ou ambos.

A manutenção de um estado de crença exato como uma fórmula lógica acaba por não ser fácil. Se houver n símbolos fluentes para o tempo t, então há 2^n estados possíveis, ou seja, atribuições de valores-verdade para aqueles símbolos. Agora, o conjunto de estados de crença é o conjunto das partes (conjunto de todos os subconjuntos) do conjunto de estados físicos. Existem 2^n estados físicos, portanto, 2^{2^n} estados de crença. Mesmo se usássemos a codificação mais compacta possível das fórmulas lógicas, com cada estado de crença representado por um único número binário, precisaríamos de números com $\log_2(2^{2^n}) = 2^n$ bits para rotular o estado de crença atual. Ou seja, a estimativa de estado exata pode exigir fórmulas lógicas cujo tamanho é exponencial no número de símbolos.

Um esquema muito comum e natural para *aproximar* a estimação de estado é representar estados de crença como conjunções de literais, ou seja, fórmulas 1-FNC. Para fazer isso, o programa agente simplesmente tenta provar X^T e $\neg X^T$ para cada símbolo X^T (assim como cada símbolo atemporal cujo valor-verdade ainda não seja conhecido), dado o estado de crença em $t - 1$. Uma conjunção de literais prováveis tornam-se o novo estado de crença, e o estado de crença anterior é descartado.

É importante entender que esse esquema pode perder alguma informação à medida que o tempo passa. Por exemplo, se a sentença na Equação 7.4 for o estado de crença verdadeiro, nem $P_{3,1}$ nem $P_{2,2}$ serão individualmente prováveis nem aparecerão no estado de crença 1-FNC (o Exercício 7.27 explora uma solução possível para esse problema). Por outro lado, devido a cada literal do estado de crença 1-FNC ser demonstrado pelo estado de crença anterior, e o estado de crença inicial ser uma afirmação verdadeira, sabemos que o estado de crença inteiro 1-FNC deve ser verdadeiro. Assim, *o conjunto de estados possíveis representado pelo estado de crença 1-FNC inclui todos os estados que são de fato possíveis, dada a história completa da percepção.* Como ilustrado na Figura 7.21, o estado de crença 1-FNC age como um simples envelope exterior, ou **aproximação conservadora**, em torno do estado de crença exato. Vemos essa ideia de aproximações conservadoras para conjuntos complicados como um tema recorrente em muitas áreas da IA.

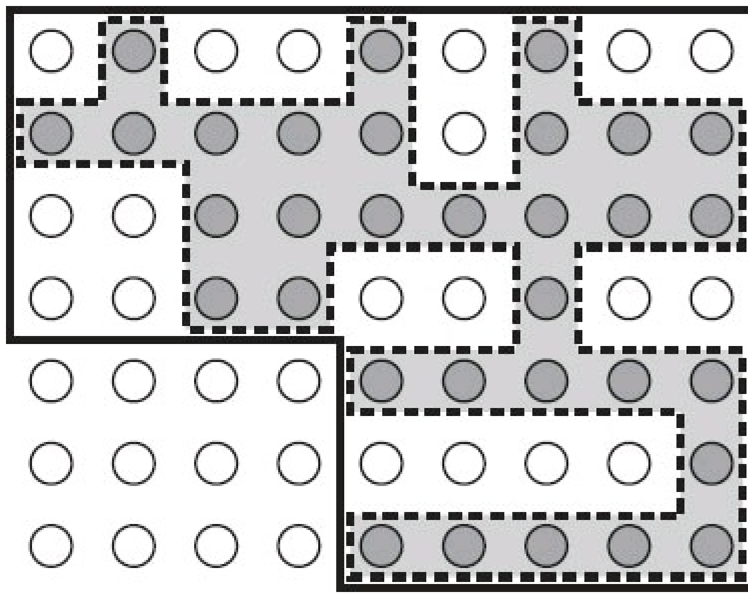


Figura 7.21 Representação de um estado de crença 1-FNC (contorno em negrito) como uma aproximação conservadora fácil de representar do exato (sinuoso) estado de crença (região sombreada com contorno tracejado). Cada mundo possível é mostrado como um círculo; os sombreados são consistentes com todas as percepções.

7.7.4 Criar planos por inferência proposicional

O agente na Figura 7.20 utiliza inferência lógica para determinar quais quadrados são seguros, mas usa uma busca A* para fazer planos. Nesta seção, vamos mostrar como fazer planos por inferência lógica. A ideia básica é muito simples:

1. Construir uma sentença que inclua:
   (a) $Init^0$, uma coleção de afirmações sobre o estado inicial;
   (b) $Transição^1, \ldots, Transição^t$, os axiomas de estado sucessor para todas as ações possíveis em cada instante até algum instante máximo t;
   (c) a afirmação de que o objetivo seja alcançado no tempo t: $TerOuro^t \land Sair^t$.
2. Apresentar a sentença toda para um resolvedor SAT. Se o resolvedor encontrar um modelo satisfazendo a sentença, o objetivo será alcançável; se a sentença é insatisfatível, o problema de planejamento é impossível.
3. Assumindo que um modelo seja encontrado, extrair do modelo as variáveis que representam ações e a que são atribuídas *verdadeiro*. Juntas, elas representam um plano para atingir os objetivos.

Na Figura 7.22 é mostrado um procedimento de planejamento proposicional, SATPLAN. Ele implementa a ideia básica recém-fornecida, com uma diferença. Devido ao agente não saber quantas etapas serão necessárias para atingir o objetivo, o algoritmo tenta cada número possível de etapas t, até algum plano máximo concebível de comprimento T_{max}. Dessa forma, é garantido o encontro do plano mais curto, se houver. Devido à forma como o SATPLAN busca uma solução, essa abordagem não pode ser usada em um ambiente parcialmente observável; o SATPLAN apenas atribuiria às

variáveis não observáveis os valores de que necessita para criar uma solução.

```
função SATPLAN(init, transição, objetivo , Tmax) retorna solução ou falha
  entradas: init, transição, objetivo, constituem uma descrição do problema
            Tmax, um limite superior para a extensão do plano

  para t = 0 até Tmax faça
    fnc ← TRADUZIR-PARA-SAT(init, transição, objetivo, t)
    modelo ← RESOLVEDOR-SAT(fnc)
    se modelo não for nulo então
      retornar EXTRAIR-SOLUÇÃO(modelo)
  retornar falha
```

Figura 7.22 O algoritmo SATPLAN. O problema de planejamento é traduzido em uma sentença FNC na qual é definido que o objetivo mantém-se em uma etapa de tempo fixo *t* e os axiomas são incluídos em cada etapa de tempo até *t*. Se o algoritmo de satisfatibilidade encontrar um modelo, é extraído um plano pela observação dos símbolos de proposição que se referem às ações e que são atribuídas como *verdadeiras* no modelo. Se não existir nenhum modelo, o processo é repetido com o objetivo movido uma etapa adiante.

A etapa-chave na utilização do SATPLAN é a construção da base de conhecimento. Pode parecer, em inspeção casual, que os axiomas do mundo de wumpus da Seção 7.7.1 sejam suficientes para as etapas 1(a) e 1(b) anteriores. Há, no entanto, uma diferença significativa entre os requisitos para a consequência lógica (como testado por ASK) e os de satisfatibilidade. Considere, por exemplo, a localização do agente, inicialmente [1,1], e suponha que objetivo pouco ambicioso do agente é estar em [2,1] no tempo 1. A base de conhecimento inicial contém $L^0_{1,1}$, e o objetivo é $L^1_{2,1}$. Utilizando ASK, podemos demonstrar $L^1_{2,1}$ se for afirmado $ParaFrente^0$, e, felizmente, não podemos demonstrar $L^1_{2,1}$ se, digamos, for afirmado $Atirar^0$, no lugar. Agora, o SATPLAN encontrará o plano [$ParaFrente^0$]; até agora, tudo certo. Infelizmente o SATPLAN também encontra o plano [$Atirar^0$]. Como pode ser? Para descobrir, inspecionamos o modelo que o SATPLAN construiu: inclui a atribuição $L^0_{2,1}$, ou seja, o agente pode estar em [2,1] no tempo 1 por estar lá no tempo 0 e atirar. Alguém poderia perguntar: “Não dissemos que o agente estava em [1,1] no tempo 0?” Sim, mas não informamos ao agente que ele não pode estar em dois lugares ao mesmo tempo! Por consequência lógica, $L^0_{2,1}$ é desconhecido e não pode, portanto, ser usado como prova; por satisfatibilidade, por outro lado, $L^0_{2,1}$ é desconhecido e pode, portanto, ser definido com qualquer valor que ajude a tornar o objetivo verdadeiro. Por essa razão, o SATPLAN é uma boa ferramenta de depuração para bases de conhecimento porque revela lugares onde o conhecimento está em falta. Nesse caso particular, podemos consertar a base de conhecimento afirmando que, em cada instante, o agente está exatamente em um local, utilizando uma coleção de sentenças semelhantes às utilizadas para afirmar a existência de exatamente um wumpus. Alternativamente, podemos afirmar $L^t_{x,y}$ para todos os outros locais que não seja [1,1]; o axioma de estado sucessor para a localização tomará conta dos instantes subsequentes. As mesmas correções também funcionam para garantir que o agente tenha apenas uma orientação.

No entanto, o SATPLAN tem mais surpresas. A primeira é que ele encontra modelos com ações

impossíveis, como atirar sem flecha. Para entender o porquê, precisamos olhar com mais cuidado ao que os axiomas de estado sucessor (como a Equação 7.3) informam sobre as ações cujas precondições não sejam satisfeitas. Os axiomas *preveem* corretamente que nada irá acontecer quando tal ação for executada (veja o Exercício 10.14), mas eles *não* dizem que a ação não pode ser executada! Para evitar a geração de planos com ações ilegais, devemos acrescentar os **axiomas de precondição** afirmando que uma ocorrência de ação requer as precondições para ser satisfeita.[13] Por exemplo, precisamos dizer, para cada instante *t*, que

$$Atirar^t \Rightarrow TemFlecha^t.$$

Isso garante que, se um plano selecionar a ação *Atirar* a qualquer tempo, o agente deverá ter uma flecha nesse momento.

A segunda surpresa do SATPLAN é a criação de planos com múltiplas ações simultâneas. Por exemplo, pode haver um modelo em que $ParaFrente^0$ e $Atirar^0$ sejam verdadeiros, o que não é permitido. Para eliminar esse problema, introduzimos os **axiomas de exclusão de ação**: para cada par de ações A_i^t e A_j^t adicionamos o axioma

$$\neg A_i^t \vee \neg A_j^t.$$

Pode ser salientado que andar para a frente e atirar ao mesmo tempo não é tão difícil de fazer, enquanto, digamos, atirar e agarrar ao mesmo tempo é impraticável. Impondo axiomas de exclusão de ação somente em pares de ações que realmente interferem umas com as outras, podemos permitir planos que incluem ações múltiplas e simultâneas — e devido ao SATPLAN encontrar o menor plano permitido, podemos ter certeza de que ele vai aproveitar essa capacidade.

Para resumir, o SATPLAN encontra modelos para uma sentença contendo o estado inicial, o objetivo, os axiomas de estado sucessor, os axiomas de precondição e os axiomas de exclusão de ação. Pode-se mostrar que esse conjunto de axiomas é suficiente, no sentido de que não há mais quaisquer “soluções” espúrias. Qualquer modelo que satisfaça a sentença proposicional será um plano válido para o problema original. A moderna tecnologia de solução SAT torna a abordagem bastante prática. Por exemplo, um resolvedor no estilo do DPLL não tem dificuldade em gerar a solução de 11 etapas para a instância do mundo de wumpus mostrado na Figura 7.2.

Esta seção descreveu uma abordagem declarativa para a construção do agente: o agente trabalha por uma combinação de sentenças afirmativas na base do conhecimento e executa inferência lógica. Essa abordagem tem algumas fraquezas escondidas em sentenças como “para cada instante *t*” e “para cada quadrado [*x*, *y*]”. Para qualquer agente prático, essas sentenças devem ser implementadas por código que automaticamente gera instâncias do esquema de sentença geral para inserção na base de conhecimento. Para um mundo de wumpus de tamanho razoável — comparável a um pequeno jogo de computador —, pode-se precisar de um tabuleiro 100 × 100 e 1.000 instantes, levando a bases de conhecimento com dezenas ou centenas de milhões de sentenças. Isso não só se torna impraticável, mas também ilustra um problema mais profundo: sabemos algo sobre o mundo de wumpus, ou seja, que a "física" funciona da mesma maneira em todas os quadrados e em todas os instantes de tempo — o que não podemos expressar diretamente na linguagem da lógica proposicional. Para resolver esse problema, precisamos de uma linguagem mais expressiva, em que sentenças como “para cada instante *t*” e “para cada quadrado [*x*, *y*]” possam ser escritas de uma forma natural. A lógica de primeira

ordem, descrita no Capítulo 8, é essa linguagem; na lógica de primeira ordem, o mundo de wumpus de qualquer tamanho e duração pode ser descrito em cerca de 10 sentenças, em vez de 10 milhões ou 10 trilhões.

7.8 RESUMO

Introduzimos o conceito de agentes baseados em conhecimento e mostramos como definir uma lógica com a qual esses agentes podem raciocinar sobre o mundo. Os principais pontos são:

- Agentes inteligentes precisam de conhecimento sobre o mundo, a fim de alcançar boas decisões.
- O conhecimento está contido em agentes sob a forma de **sentenças** em uma **linguagem de representação do conhecimento** que está armazenada em uma **base de conhecimento**.
- Um agente baseado em conhecimento é composto por uma base de conhecimento e um mecanismo de inferência. Ele opera armazenando sentenças sobre o mundo em sua base de conhecimento, utilizando o mecanismo de inferência para deduzir novas sentenças e empregando essas sentenças para decidir que ação executar.
- Uma linguagem de representação é definida por sua **sintaxe**, que especifica a estrutura de sentenças, e por sua **semântica**, que define a **verdade** de cada sentença em cada **modelo** ou **mundo possível**.
- A relação de **consequência lógica** entre sentenças é crucial para nossa compreensão do raciocínio. Uma sentença a tem como consequência lógica outra sentença β se β é verdadeira em todos os mundos em que a é verdadeira. Definições equivalentes incluem a **validade** da sentença a $\Rightarrow \beta$ e a **não satisfatibilidade** da sentença a $\wedge \neg\beta$.
- A inferência é o processo de derivação de novas sentenças a partir de sentenças antigas. Os algoritmos de inferência **corretos** derivam *somente* consequências lógicas; os algoritmos **completos** derivam *todas* as consequências lógicas.
- A **lógica proposicional** é uma linguagem muito simples que consiste em **símbolos de proposições** e **conectivos lógicos**. Ela pode manipular proposições que são conhecidas como verdadeiras, conhecidas como falsas ou completamente desconhecidas.
- O conjunto de modelos possíveis, dado um vocabulário proposicional fixo, é finito e, assim, a consequência lógica pode ser verificada pela enumeração de modelos. Algoritmos de inferência eficientes de **verificação de modelos** para lógica proposicional incluem métodos de retrocesso e busca local e, com frequência, podem resolver grandes problemas com muita rapidez.
- **Regras de inferência** são padrões de inferência consistente que podem ser usados para descobrir provas. A **regra de resolução** gera um algoritmo de inferência completo para bases de conhecimento expressas em **forma normal conjuntiva**. O **encadeamento para a frente** e o **encadeamento para trás** são algoritmos de raciocínio muito naturais para bases de conhecimento na **forma de Horn**.
- Métodos de **busca local**, tal como o WALKSAT, podem ser utilizados para encontrar soluções. Tais algoritmos são consistentes, mas não completos.
- A **estimação de estado** lógico envolve a manutenção de uma sentença lógica que descreve o conjunto de estados possíveis coerentes com o histórico de observação. Cada etapa de

atualização exige inferência utilizando o modelo de transição do ambiente, que é construído de **axiomas de estados sucessores** que especificam como cada **fluente** se altera.

- As decisões dentro de um agente lógico podem ser realizadas por resolução SAT: encontrar modelos possíveis especificando sequências de ações futuras que atingem a meta. Essa abordagem funciona apenas para ambientes totalmente observáveis ou sem sensores.
- A lógica proposicional não consegue tratar de ambientes de tamanho ilimitado porque lhe falta a capacidade de expressão necessária para lidar de forma concisa com tempo, espaço e padrões universais de relacionamentos entre objetos.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O artigo de John McCarthy “Programs with Common Sense” (McCarthy, 1958, 1968) promulgou a noção de agentes que utilizam raciocínio lógico para atuar como mediadores entre percepções e ações. Ele também levantou a bandeira do declarativismo, assinalando que informar a um agente o que ele precisa saber é uma forma muito elegante de criar software. O artigo de Allen Newell (1982), “The Knowledge Level”, argumenta que agentes racionais podem ser descritos e analisados em um nível abstrato definido pelo conhecimento que possuem, e não pelos programas que executam. As abordagens declarativa e procedural para a IA são comparadas em Boden (1977). O debate foi reavivado por Brooks (1991) e Nilsson (1991) entre outros, e continua até hoje (Shaparau *et al*., 2008). Enquanto isso, a abordagem declarativa se espalhou para outras áreas da ciência da computação, tais como redes (Loo *et al*., 2006).

A lógica em si teve suas origens na filosofia e na matemática dos antigos gregos. Vários princípios lógicos — os princípios que conectam a estrutura sintática de sentenças com sua verdade e falsidade, com seu significado ou com a validade de argumentos em que elas figuram — foram difundidos nos trabalhos de Platão. O primeiro estudo sistemático conhecido de lógica foi desenvolvido por Aristóteles, cujo trabalho foi reunido por seus alunos após sua morte, em 322 a.C., sob a forma de um tratado denominado *Organon*. Os **silogismos** de Aristóteles eram aquilo que chamaríamos agora de regras de inferência. Embora os silogismos incluíssem elementos tanto de lógica proposicional quanto de lógica de primeira ordem, faltaram no sistema como um todo as propriedades de composição necessárias para lidar com as sentenças de complexidade arbitrária.

As escolas intimamente relacionadas megárica e estoica (originárias do século V a.C. e que continuaram por vários séculos daí em diante) começaram o estudo sistemático dos conectivos lógicos básicos. O uso de tabelas--verdade para definir conectivos lógicos se deve a Filo de Mégara. Os estoicos tomavam cinco regras básicas de inferência como válidas sem prova, inclusive a regra agora denominada *Modus Ponens*. Eles derivaram muitas outras regras a partir dessas cinco, utilizando entre outros princípios o teorema da dedução (Seção 7.5) e eram muito mais claros em relação à noção de prova do que Aristóteles. Um bom relato da história da lógica megárica e estoica, até onde ela é conhecida, é apresentado por Benson Mates (1953).

A ideia de reduzir a inferência lógica a um processo puramente mecânico aplicado a uma linguagem formal se deve a Wilhelm Leibniz (1646-1716), apesar de ele ter tido sucesso limitado na implementação das ideias.

George Boole (1847) introduziu o primeiro sistema completo e funcional de lógica formal em seu livro *The Mathematical Analysis of Logic*. A lógica de Boole era fortemente modelada sobre a álgebra comum de números reais e utilizava a substituição de expressões logicamente equivalentes como seu principal método de inferência. Embora o sistema de Boole ainda estivesse abaixo da lógica proposicional completa, ele se encontrava próximo o suficiente dessa lógica para que outros matemáticos pudessem preencher rapidamente as lacunas. Schröder (1877) descreveu a forma normal conjuntiva, enquanto a forma de Horn foi introduzida muito mais tarde por Alfred Horn (1951). A primeira exposição completa da lógica proposicional moderna (e da lógica de primeira ordem) é encontrada no trabalho de Gottlob Frege (1879), *Begriffschrift* ("Escrita de conceitos" ou "Notação conceitual").

O primeiro dispositivo mecânico destinado a executar inferências lógicas foi construído pelo terceiro Conde de Stanhope (1753-1816). O Demonstrador de Stanhope podia tratar silogismos e certas inferências na teoria de probabilidade. William Stanley Jevons, um dos cientistas que aperfeiçoaram e estenderam o trabalho de Boole, construiu seu "piano lógico" em 1869 para executar inferências em lógica booleana. Uma divertida e instrutiva história desses e de outros antigos dispositivos mecânicos para raciocínio é apresentada por Martin Gardner (1968). O primeiro programa de computador publicado para inferência lógica foi o Logic Theorist de Newell, Shaw e Simon (1957). Esse programa foi planejado para modelar processos humanos de pensamento. Martin Davis (1957) realmente projetou um programa que apresentou uma prova em 1954, mas os resultados do Logic Theorist foram publicados um pouco antes.

As tabelas-verdade como método de teste da validade ou da não satisfatibilidade de sentenças na linguagem da lógica proposicional foram introduzidas independentemente por Emil Post (1921) e Ludwig Wittgenstein (1922). Na década de 1930, foram feitos muitos progressos em métodos de inferência para lógica de primeira ordem. Em particular, Gödel (1930) mostrou que um procedimento completo para inferência em lógica de primeira ordem podia ser obtido por meio de uma redução à lógica proposicional, com a utilização do teorema de Herbrand (Herbrand, 1930). Veremos novamente essa história no Capítulo 9; aqui, é importante ressaltar que o desenvolvimento de algoritmos proposicionais eficientes nos anos 1960 foi motivado em grande parte pelo interesse dos matemáticos em um provador de teoremas eficaz para lógica de primeira ordem. O algoritmo de Davis-Putnam (Davis e Putnam, 1960) foi o primeiro algoritmo efetivo para resolução proposicional, mas, na maioria dos casos, era muito menos eficiente que o algoritmo de retrocesso de DPLL introduzido dois anos mais tarde (1962). A regra de resolução completa e uma prova de sua completude apareceram em um importante artigo de J. A. Robinson (1965), que também mostrou como realizar o raciocínio de primeira ordem sem recorrer a técnicas proposicionais.

Stephen Cook (1971) mostrou que o problema de decidir a satisfatibilidade de uma sentença em lógica proposicional (o problema SAT) é NP-completo. Tendo em vista que decidir a consequência lógica é equivalente a decidir a não satisfatibilidade, o problema é co-NP-completo. São conhecidos muitos subconjuntos da lógica proposicional para os quais o problema de satisfatibilidade é resolvível em tempo polinomial; as cláusulas de Horn formam um desses subconjuntos. O algoritmo de encadeamento para a frente em tempo linear para cláusulas de Horn se deve a Dowling e Gallier (1984), que descrevem seu algoritmo como um processo de fluxo de dados semelhante à propagação de sinais em um circuito.

As primeiras investigações teóricas mostraram que o DPLL tem complexidade polinomial no caso médio para certas distribuições naturais de problemas. Esse fato potencialmente interessante se tornou menos interessante quando Franco e Paull (1983) mostraram que os mesmos problemas podiam ser resolvidos em tempo constante, simplesmente supondo-se atribuições aleatórias. O método de geração aleatória descrito no capítulo gera problemas muito mais difíceis. Motivado pelo sucesso empírico da busca local nesses problemas, Koutsoupias e Papadimitriou (1992) mostraram que um simples algoritmo de subida de encosta pode resolver *quase todas* as instâncias de problemas de satisfatibilidade com muita rapidez, sugerindo que problemas difíceis são raros. Além disso, Schöning (1999) exibiu um algoritmo de subida de encosta aleatório que apresentava um tempo de execução esperado no *pior caso* de execução nos problemas 3-SAT (isto é, a satisfatibilidade das sentenças 3-FNC) de $O(1{,}333^n)$ — ainda exponencial, embora substancialmente mais rápido que os limites anteriores de pior caso. O recorde atual é $O(1{,}324^n)$ (Iwama e Tamaki, 2004). Achlioptas *et al*. (2004) e Alekhnovich *et al*. (2005) apresentaram famílias de instâncias de 3-SAT para o qual todos os algoritmos como o DPLL requerem tempo de execução exponencial.

Pelo lado prático, ganhos de eficiência têm sido obtidos com resolvedores proposicionais. Dados 10 minutos de tempo de computação, o algoritmo DPLL original de 1962 só podia resolver problemas com não mais do que 10 ou 15 variáveis. Em 1995, o solucionador Satz (Li e Anbulagan, 1997) podia lidar com mil variáveis, graças às estruturas de dados otimizadas de indexação de variáveis. Duas contribuições fundamentais foram a técnica de indexação de **literal vigiado** de Zhang e Stickel (1996), que tornou a propagação unitária muito eficiente, e a introdução das técnicas de aprendizagem de cláusulas (ou seja, restrição) da comunidade PSR por Bayardo e Schrag (1997). Utilizando essas ideias, e estimulados pela perspectiva de solução de problemas de verificação de circuito em escala industrial, Moskewicz *et al*. (2001) desenvolveram o resolvedor CHAFF, que podia lidar com problemas com milhões de variáveis. A partir de 2002, as competições SAT foram realizadas regularmente; a maioria dos vencedores eram descendentes de CHAFF ou utilizavam a mesma abordagem geral. O RSAT (Pipatsrisawat e Darwiche, 2007), o vencedor de 2007, entrou na última categoria. Também digno de nota é o MINISAT (Een e Sorensson, 2003), uma implementação open-source disponível em http://minisat.se, que foi projetada para ser facilmente modificada e melhorada. O cenário atual dos resolvedores foi examinado por Gomes *et al.* (2008).

Os algoritmos de busca local para satisfatibilidade foram experimentados por vários autores em toda a década de 1980; todos os algoritmos foram baseados na ideia de minimizar o número de cláusulas não satisfeitas (Hansen e Jaumard, 1990). Um algoritmo particularmente eficaz foi desenvolvido por Gu (1989) e de forma independente por Selman *et al.* (1992), que o chamaram de GSAT e mostraram que ele era capaz de resolver uma ampla gama de problemas difíceis muito rapidamente. O algoritmo WALKSAT descrito no capítulo é devido a Selman *et al*. (1996).

A "transição de fase" em satisfatibilidade de problemas *k*-SAT aleatórios foi observada pela primeira vez por Simon e Dubois (1989) e deu origem a uma grande quantidade de pesquisas teóricas e empíricas — devida, em parte, à óbvia ligação com o fenômeno da transição de fase em física estatística. Cheeseman *et al.* (1991) observaram transições de fase em vários PSRs e conjecturaram que todos os problemas NP-difíceis têm uma fase de transição. Crawford e Auton (1993) localizaram a transição 3-SAT em uma proporção cláusula/variável de cerca de 4,26, observando que isso coincide com picos acentuados no tempo de execução de seu resolvedor SAT. Cook e Mitchell

(1997) forneceram um resumo excelente de literatura anterior sobre o problema.

O estado atual do conhecimento teórico foi resumido por Achlioptas (2009). A **conjectura do limiar da satisfatibilidade** afirma que, para cada k, há um limiar nítido de satisfatibilidade r_k, tal que quando o número de variáveis $n \to \infty$, as instâncias abaixo do limiar são *satisfatíveis* com probabilidade 1, enquanto aquelas acima do limiar são *insatisfatíveis* com probabilidade 1. A suposição não foi provada completamente por Friedgut (1999): existe um limiar pronunciado, mas sua localização pode depender de n, mesmo quando $n \to \infty$. Apesar dos progressos significativos em análise assintótica da localização do limite para k grande (Achlioptas e Peres, 2004; Achlioptas *et al*., 2007), tudo o que pode ser provado para $k = 3$ é que ele está no intervalo [3,52; 4,51]. A teoria atual sugere que um pico no tempo de execução de um solucionador SAT não está necessariamente relacionado com o limiar da satisfatibilidade, mas com uma transição de fase na distribuição de solução e estrutura das instâncias de SAT. Os resultados empíricos, devidoa a Coarfa *et al*. (2003) sustentam essa visão. De fato, os algoritmos tais como **propagação de inspeção** (Parisi e Zecchina, 2002; Maneva *et al*., 2007) aproveitam propriedades especiais de instâncias SAT aleatórias próximas ao limiar da satisfatibilidade e superam em muito os solucionadores SAT genéricos em tais instâncias.

As melhores fontes de informação sobre satisfatibilidade, tanto teóricas como práticas, são o *Handbook of Satisfiability* (Biere *et al*., 2009) e as *International Conferences on Theory and Applications of Satisfiability Testing* regulares, conhecidas como SAT.

A ideia de construir agentes com lógica proposicional pode ser rastreada até o trabalho seminal de McCulloch e Pitts (1943), que iniciou o campo das redes neurais. Ao contrário da suposição popular, o documento estava preocupado com a implementação de um projeto de agente baseado em circuito booleano no cérebro. No entanto, agentes baseados em circuitos, que realizam a computação por propagação de sinais nos circuitos de hardware em vez de executar algoritmos em computadores de uso geral, receberam pouca atenção em IA. A exceção mais notável foi o trabalho de Stan Rosenschein (Rosenschein, 1985; Kaelbling e Rosenschein, 1990), que desenvolveu maneiras para compilar agentes baseados em circuito a partir de descrições declarativas do ambiente da tarefa (a abordagem de Rosenschein foi descrita com algum pormenor na segunda edição deste livro). O trabalho de Rod Brooks (1986, 1989) demonstra a eficácia dos projetos baseados em circuito para controlar robôs — um tema que ocupa o Capítulo 25. Brooks (1991) argumenta que os projetos com base em circuito são *tudo* o que é necessário para IA — que representação e raciocínio são pesados, caros e desnecessários. Em nossa opinião, nenhuma das abordagens é suficiente por si só. Williams *et al*. (2003) mostram como um projeto de agente híbrido não muito diferente do nosso agente de wumpus tem sido utilizado para controlar a nave espacial da Nasa, o planejamento de sequências de ações e o diagnóstico e recuperação das falhas.

O problema geral de manter o controle de um ambiente parcialmente observável foi introduzido para representações baseadas em estado no Capítulo 4. Sua instanciação para as representações proposicionais foi estudada por Amir e Russell (2003), que identificaram várias classes de ambientes que admitem algoritmos de estimação de estado eficientes e mostraram que, para várias outras classes, o problema é intratável. O problema de **projeção temporal**, que envolve determinar que proposições se mantêm verdadeiras depois que uma sequência de ações foi executada, pode ser visto como um caso especial de estimação de estado com percepção vazia. Muitos autores estudaram

esse problema devido à sua importância em planejamento; alguns resultados sobre complexidade importantes foram estabelecidos por Liberatore (1997). A ideia de representar um estado de crença com proposições pode ser atribuída a Wittgenstein (1922).

A estimação do estado lógico, é claro, exige uma representação lógica dos efeitos das ações — um dos principais problemas em IA desde o final dos anos 1950. A proposta dominante tem sido o formalismo do **cálculo de situações** (McCarthy, 1963), que está expresso dentro da lógica de primeira ordem. Discutiremos o cálculo de situações e várias extensões e alternativas nos Capítulos 10 e 12. A abordagem adotada neste capítulo — utilizando índices temporais sobre variáveis proposicionais — é mais restritiva, mas tem a vantagem da simplicidade. A abordagem geral, personificada no algoritmo SATPLAN, foi proposta por Kautz e Selman (1992). Gerações posteriores de SATPLAN foram capazes de tirar vantagem dos avanços em solucionadores SAT, descritos anteriormente, e se manter entre as formas mais eficazes de resolver problemas difíceis (Kautz, 2006).

O **problema do persistência** foi reconhecido pela primeira vez por McCarthy e Hayes (1969). Muitos pesquisadores consideraram o problema insolúvel dentro da lógica de primeira ordem e isso gerou uma grande quantidade de pesquisa em lógicas não monotônicas. Filósofos, de Dreyfus (1972) a Crockett (1994), citaram o problema de persistência como um sintoma da falha inevitável de toda iniciativa em IA. A solução do problema de persistência com axiomas de estado sucessor é devida a Ray Reiter (1991). Thielscher (1999) identificou o problema de persistência inferencial como uma ideia separada e forneceu uma solução. Em retrospecto, pode-se observar que os agentes de Rosenschein (1985) utilizavam circuitos que implementaram axiomas de estado sucessor, mas Rosenschein não percebeu que com isso o problema de persistência estava em grande parte resolvido. Foo (2001) explicou por que os modelos de teoria de controle de evento discreto, normalmente utilizados pelos engenheiros, não têm que lidar explicitamente com o problema de persistência: porque eles estão lidando com previsão e controle, não com explicação e raciocínio sobre situações contrafactuais.

Os resolvedores proposicionais modernos têm ampla aplicabilidade em aplicações industriais. A aplicação de inferência proposicional na síntese de hardware do computador é agora uma técnica-padrão com muitas implementações de grande escala (Nowick *et al*., 1993). O verificador de satisfatibilidade SATMC foi utilizado para detectar uma vulnerabilidade até então desconhecida em um protocolo de entrada de usuário de navegador da Web (Armando *et al*., 2008).

O mundo de wumpus foi inventado por Gregory Yob (1975). Ironicamente, Yob o desenvolveu porque estava entediado com jogos que se desenrolavam em tabuleiro retangular: a topologia de seu mundo de wumpus original era um dodecaedro, e nós a colocamos de volta no velho e incômodo tabuleiro. Michael Genesereth foi o primeiro a sugerir que o mundo de wumpus fosse usado como uma plataforma de testes de agentes.

EXERCÍCIOS

7.1 Suponha que o agente tenha progredido até o ponto mostrado na Figura 7.4(a), tendo percebido nada em [1,1], uma brisa em [2,1] e um fedor em [1,2], e agora está preocupado com o conteúdo de

[1,3], [2,2] e [3,1]. Cada uma dessas posições pode conter um poço e, no máximo, uma pode conter um wumpus. Seguindo o exemplo da Figura 7.5, construa o conjunto de mundos possíveis (você deve encontrar 32 deles). Marque os mundos em que a BC é verdadeira e aqueles em que cada uma das sentenças a seguir é verdadeira:

α_2 = "Não existe nenhum poço em [2,2]".
α_3 = "Existe um wumpus em [1,3]".

Consequentemente, mostre que $BC \models \alpha_2$ e $BC \models \alpha_3$.

7.2 (Adaptado de Barwise e Etchemendy (1993).) Dada a sentença a seguir, você poderia demonstrar que o unicórnio é mítico? E que é mágico? E que tem chifre?

Se o unicórnio é mítico, então é imortal; porém, se ele não é mítico, então é um mamífero mortal. Se o unicórnio é imortal ou um mamífero, então ele tem chifre. O unicórnio é mágico se tem chifre.

7.3 Considere o problema de decidir se uma sentença de lógica proposicional é verdadeira em determinado modelo.

a. Escreva um algoritmo recursivo VERDADEIRO-LP?(s, m) que retorne *verdadeiro* se e somente se a sentença s for verdadeira no modelo m (onde m atribui um valor-verdade a todo símbolo em s). O algoritmo deve funcionar em tempo linear em relação ao tamanho da sentença. (Como alternativa, use uma versão dessa função obtida no repositório de código on-line.)

b. Forneça três exemplos de sentenças que possam ser definidas como verdadeiras ou falsas em um modelo *parcial* que não especifica um valor-verdade para alguns dos símbolos.

c. Mostre que o valor-verdade (se houver) de uma sentença em um modelo parcial não pode ser determinado de maneira eficiente no caso geral.

d. Modifique seu algoritmo VERDADEIRO-LP? de forma que às vezes ele possa julgar a verdade a partir de modelos parciais, ao mesmo tempo em que mantém sua estrutura recursiva e seu tempo de execução linear. Forneça três exemplos de sentenças cuja verdade em um modelo parcial *não* seja detectada pelo seu algoritmo.

e. Investigue se o algoritmo modificado torna CONSEQUÊNCIA-LÓGICA-TV? mais eficiente.

7.4 Qual dos seguintes está correto?

a. *Falso* $\models$ *Verdadeiro*.

b. *Verdadeiro* $\models$ *Falso*.

c. $(A \wedge B) \models (A \Leftrightarrow B)$.

d. $A \Leftrightarrow B \models A \vee B$.

e. $A \Leftrightarrow B \models \neg A \vee B$.

f. $(A \wedge B) \Rightarrow C \models (A \Rightarrow C) \vee (B \Rightarrow C)$.

g. $(C \vee (\neg A \wedge \neg B)) \equiv ((A \Rightarrow C) \wedge (B \Rightarrow C))$.

h. $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B)$.

i. $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B) \wedge (\neg D \vee E)$.

j. $(A \vee B) \wedge \neg(A \Rightarrow B)$ é satisfatível.

k. $(A \Leftrightarrow B) \wedge (\neg A \vee B)$ é satisfatível.

l. $(A \Leftrightarrow B) \Leftrightarrow C$ tem o mesmo número de modelos que $(A \Leftrightarrow B)$ para qualquer conjunto de símbolos de proposição fixos que inclui A, B, C.

7.5 Demonstre cada uma das asserções a seguir:

a. α é válida se e somente se *Verdadeiro* $\models \alpha$.

b. Para qualquer α, *Falso* $\models \alpha$.

c. $\alpha \models \beta$ se e somente se a sentença $(\alpha \Rightarrow \beta)$ é válida.

d. $\alpha \equiv \beta$ se e somente se a sentença $(\alpha \Leftrightarrow \beta)$ é válida.

e. $\alpha \models \beta$ se e somente se a sentença $(\alpha \wedge \neg\beta)$ é não satisfatível.

7.6 Demonstre ou encontre um contraexemplo para cada uma das seguintes asserções:

a. Se $\alpha \models \gamma$ ou $\beta \models \gamma$ (ou ambos) então $(\alpha \wedge \beta) \models \gamma$

b. Se $\alpha \models (\beta \wedge \gamma)$ então $\alpha \models \beta$ e $\alpha \models \gamma$.

c. Se $\alpha \models (\beta \vee \gamma)$ então $\alpha \models \beta$ ou $\alpha \models \gamma$ (ou ambos).

7.7 Considere um vocabulário com apenas quatro proposições, *A*, *B*, *C* e *D*. Quantos modelos existem para as sentenças a seguir?

a. $B \vee C$.

b. $\neg A \vee \neg B \vee \neg C \vee \neg D$.

c. $(A \Rightarrow B) \wedge A \wedge \neg B \wedge C \wedge D$.

7.8 Definimos quatro conectivos lógicos binários diferentes.

a. Existem outros que possam ser úteis?

b. Quantos conectivos binários podem existir?

c. Por que alguns deles não são muito úteis?

7.9 Usando um método de sua escolha, verifique cada uma das equivalências da Figura 7.11.

7.10 Decida se cada uma das sentenças a seguir é válida, não satisfatível ou nenhuma dessas opções. Verifique suas decisões usando tabelas-verdade ou as regras de equivalência da Figura 7.11.

a. *Fumaça* $\Rightarrow$ *Fumaça*

b. *Fumaça* $\Rightarrow$ *Fogo*

c. (*Fumaça* $\Rightarrow$ *Fogo*) $\Rightarrow$ ($\neg$*Fumaça* $\Rightarrow$ $\neg$*Fogo*)

d. *Fumaça* $\vee$ *Fogo* $\vee$ $\neg$*Fogo*

e. $((Fumaça \wedge Calor) \Rightarrow Fogo) \Leftrightarrow ((Fumaça \Rightarrow Fogo) \vee (Calor \Rightarrow Fogo))$

f. $(Fumaça \Rightarrow Fogo) \Rightarrow ((Fumaça \wedge Calor) \Rightarrow Fogo)$

g. $Gande \vee Burro \vee (Grande \Rightarrow Burro)$

7.11 Qualquer sentença lógica proposicional é logicamente equivalente à asserção de que cada mundo possível em que ela seria falsa não deveria ocorrer. A partir dessa observação, prove que qualquer sentença pode ser escrita em FNC.

7.12 Utilize resolução para demonstrar a sentença $\neg A \wedge \neg B$ das cláusulas no Exercício 7.20.

7.13 Este exercício examina o relacionamento entre cláusulas e sentenças de implicação.

a. Mostre que a cláusula $(\neg P_1 \vee \ldots \vee \neg P_m \vee Q)$ é logicamente equivalente à sentença de implicação $(P_1 \wedge \ldots \wedge P_m) \Rightarrow Q$.

b. Mostre que toda cláusula (independentemente do número de literais positivos) pode ser escrita na forma $(P_1 \wedge \ldots \wedge P_m) \Rightarrow (Q_1 \vee \ldots \vee Q_n)$, onde os valores de P e Q são símbolos de proposições. Uma base de conhecimento que consiste em tais sentenças está em **forma normal implicativa** ou **forma de Kowalski** (Kowalski, 1979).

c. Escreva a regra de resolução completa para sentenças em forma normal implicativa.

7.14 De acordo com alguns especialistas políticos, uma pessoa que é radical (R) é elegível (E) se for conservadora (C), mas de oura forma não será elegível.

a. Quais das seguintes são representações corretas dessa afirmação?

(i) $(R \wedge E) \Leftrightarrow C$

(ii) $R \Rightarrow (E \Leftrightarrow (C)$

(iii) $R \Rightarrow ((C \Rightarrow E) \vee \neg E)$

b. Quais das sentenças em (a) podem ser expressas na forma de Horn?

7.15 Esta questão considera a representação dos problemas de satisfatibilidade (SAT) como PSRs.

a. Desenhe o grafo de restrição correspondente ao problema SAT

$$(\neg X_1 \vee X_2) \wedge (\neg X_2 \vee X_3) \wedge \ldots \wedge (\neg X_{n-1} \vee X_n)$$

para o caso particular de $n = 5$.

b. Quantas soluções existem para este problema SAT geral como função de n?

c. Suponha que apliquemos a BUSCA-POR-RETROCESSO para encontrar *todas* as soluções para um SAT PSR do tipo dado em (a). (Para encontrar todas as soluções para um PSR, simplesmente modificamos o algoritmo básico de modo que ele continue a busca depois de encontrar cada solução.) Suponha que as variáveis sejam ordenadas de $X_1, \ldots, X_n$ e *falso* é ordenado antes de *verdadeiro*. Quanto tempo levará o algoritmo para terminar? (Escreva uma expressão $O(\cdot)$ como função de n.)

d. Sabemos que os problemas SAT na forma de Horn podem ser resolvidos em tempo linear por encadeamento para a frente (propagação unitária). Sabemos também que todos os PSRs binários

estruturados em árvore com domínios discretos, finitos, podem ser resolvidos em tempo linear pelo número de variáveis (Seção 6.5). Existe ligação entre esses dois fatos? Discuta.

7.16 Explique por que toda a cláusula proposicional não vazia, por sí só, é satisfatível. Prove acuradamente que cada conjunto de cinco cláusulas 3-SAT é satisfatível, dado que cada cláusula menciona exatamente três variáveis distintas. Qual o menor conjunto dessas cláusulas que é insatisfatível? Construa tal conjunto.

7.17 Uma expressão proposicional 2-FNC é uma conjunção de cláusulas, cada uma contendo exatamente 2 literais, isto é,

$$(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee D) \wedge (\neg C \vee G) \wedge (\neg D \vee G).$$

a. Utilizando resolução demonstre que a sentença acima tem como consequência lógica G.

b. Duas cláusulas são *semanticamente distintas* se não forem logicamente equivalentes. Quantas cláusulas 2-FNC semanticamente distintas podem ser construídas de n símbolos proposicionais?

c. Usando a resposta ao item (b), demonstre que a resolução proposicional sempre termina em tempo polinomial em n dada uma sentença 2-FNC contendo não mais que n símbolos distintos.

d. Explique por que o argumento no item (c) não se aplica a 3-FNC.

7.18 Considere a seguinte sentença:

$$[(Comida \Rightarrow Festa) \vee (Bebidas \Rightarrow Festa)] \Rightarrow [(Comida \wedge Bebidas) \Rightarrow Festa].$$

a. Determine, utilizando enumeração, se essa sentença é válida, satisfatível (mas não válida), insatisfatível.

b. Converta os lados esquerdo e direito da implicação principal em FNC, mostrando cada etapa, e explique como os resultados confirmam a sua resposta para (a).

c. Demonstre a sua resposta para (a) utilizando resolução.

7.19 Uma sentença está na **forma normal disjuntiva** (FND), se for uma disjunção de conjunções de literais. Por exemplo, a sentença $(A \wedge B \wedge \neg C) \vee (\neg A \wedge C) \vee (B \wedge \neg C)$ está em FND.

a. Qualquer sentença lógica proposicional é logicamente equivalente à afirmação de que algum mundo possível no qual a sentença seria verdadeira é de fato o caso. A partir dessa observação, demonstre que qualquer sentença pode ser escrita em FND.

b. Construa um algoritmo que converta qualquer sentença de lógica proposicional em FND. (*Dica*: O algoritmo é similar ao algoritmo de conversão para FNC dado na Seção 7.5.2.)

c. Construa um algoritmo simples que tome como entrada uma sentença em FND e retorne uma valoração que a satisfaz, se existir, ou relate que nenhuma valoração satisfatória existe.

d. Aplique os algoritmos em (b) e (c) para o seguinte conjunto de sentenças:

$A \Rightarrow B$

$B \Rightarrow C$

$C \Rightarrow \neg A.$

e. Como o algoritmo em (b) é muito semelhante ao algoritmo de conversão para FNC, e como o

algoritmo em (c) é muito mais simples do que qualquer algoritmo para resolver um conjunto de sentenças em FNC, por que essa técnica não é utilizada no raciocínio automatizado?

7.20 Converta o seguinte conjunto de sentenças para a forma clausal.

S1: $A \Leftrightarrow (B \lor E)$.

S2: $E \Rightarrow D$.

S3: $C \land F \Rightarrow \neg B$.

S4: $E \Rightarrow B$.

S5: $B \Rightarrow F$.

S6: $B \Rightarrow C$.

Apresente um rastreamento (trace) da execução de DPLL sobre o conjunto dessas cláusulas.

7.21 Uma sentença 4-FNC gerada randomicamente com n símbolos e m cláusulas é mais ou menos provável de ser resolvida do que uma sentença 3-FNC gerada aleatoriamente com n símbolos e m cláusulas? Explique.

7.22 Campo Minado, o conhecido jogo de computador, está intimamente relacionado ao mundo de wumpus. Um mundo de campo minado é uma malha retangular de N quadrados com M minas invisíveis espalhadas entre eles. Qualquer quadrado pode ser sondado pelo agente; se uma mina for sondada, segue-se a morte imediata. Campo Minado indica a presença de minas, revelando, em cada quadrado sondado, o *número* de minas adjacentes diretamente ou em diagonal. O objetivo é sondar todo quadrado não minado.

a. Seja $X_{i,j}$ verdadeira se e somente se o quadrado $[i, j]$ contém uma mina. Anote a asserção de que exatamente duas minas são adjacentes a [1,1] como uma sentença envolvendo alguma combinação lógica de $X_{i,j}$ proposições.

b. Generalize sua asserção de (a) explicando como construir uma sentença FNC afirmando que k dentre n vizinhos contém minas.

c. Explique exatamente como um agente pode usar DPLL para provar que determinado quadrado contém (ou não contém) uma mina, ignorando a restrição global de que existem exatamente M minas ao todo.

d. Suponha que a restrição global seja construída por meio do seu método da parte (b). De que maneira o número de cláusulas depende de M e N? Sugira um modo de modificar DPLL para que a restrição global não precise ser representada de forma explícita.

e. Alguma conclusão derivada pelo método da parte (c) é invalidada quando a restrição global é levada em consideração?

f. Forneça exemplos de configurações de valores de sondagem que induzem *dependência de longo prazo* tais que o conteúdo de um dado quadrado não sondado forneceria informações sobre o conteúdo de um quadrado muito distante. [*Sugestão*: Considere um tabuleiro de $N \times 1$.]

7.23 Quanto tempo demora para provar que $BC \models \alpha$ usando DPLL quando α é um literal *já contido em BC*? Explique.

7.24 Descreva o comportamento de DPLL na base de conhecimento da Figura 7.16 quando se tenta provar *Q* e compare esse comportamento com o do algoritmo de encadeamento para a frente.

7.25 Escreva um axioma de estado sucessor para o predicado *trancado* que se aplica a portas, assumindo que as únicas ações disponíveis são *trancado* e *destrancado*.

7.26 A Seção 7.7.1 fornece alguns dos axiomas de estado sucessor necessários para o mundo de wumpus. Escreva os axiomas para todos os símbolos fluentes restantes.

7.27 Modifique o AGENTE-WUMPUS-HÍBRIDO para utilizar o método de estimação de estado lógico por 1-FCN (Seção 7.7.3). Naquela seção observamos que tal agente não será capaz de adquirir, manter e utilizar crenças mais complexas como a da disjunção $P_{3,1} \lor P_{2,2}$. Sugira um método para superar esse problema através da definição de símbolos de proposição adicionais e os experimente no mundo de wumpus. Isso melhorará o desempenho do agente?

---

[1] **Lógica difusa**, discutida no Capítulo 14, permite graus de verdade.

[2] Embora a figura mostre os modelos como mundos de wumpus parciais, na realidade eles não são nada além de atribuições de valores *verdadeiro* e *falso* às sentenças "existe um poço em [1,2]" etc. Os modelos, em sentido matemático, não precisam conter "horríveis monstros cabeludos".

[3] O agente pode calcular a *probabilidade* de existir um poço em [2,2]; o Capítulo 13 mostra como fazê-lo.

[4] A verificação de modelos funciona se o espaço de modelos é finito — por exemplo, em mundos de wumpus de tamanho fixo. Por outro lado, no caso da aritmética, o espaço de modelos é infinito: até mesmo se nos limitarmos aos inteiros, haverá infinitamente muitos pares de valores para x e y na sentença $x + y = 4$.

[5] Compare com o caso dos espaços de busca infinitos do Capítulo 3, em que a busca em profundidade não é completa.

[6] Como define Wittgenstein (1922) em seu famosos *Tractatus*: "O mundo é tudo que é o caso."

[7] O latim tem uma palavra diferente, *aut*, para indicar o ou exclusivo.

[8] As lógicas **não monotônicas**, que violam a propriedade de monotonicidade, captam uma propriedade comum do raciocínio humano: a mudança de ideia. Elas são discutidas na Seção 12.6.

[9] Se uma cláusula for vista como um *conjunto* de literais, essa restrição será automaticamente respeitada. O uso da notação de conjuntos para cláusulas torna a regra de resolução muito mais clara, ao custo de introduzir uma notação adicional.

[10] A Seção 7.4.3 explicou convenientemente sobre esse requisito.

[11] O nome "problema de persistência" vem de "persistência de referência" em física — o segundo plano estacionário assumido em relação ao movimento que está sendo medido. Ele também faz analogia com os quadros (persistências) de um filme, em que normalmente a maior parte do segundo plano permanece constante, enquanto as mudanças ocorrem em primeiro plano.

[12] Podemos pensar sobre a própria história da percepção como uma representação do estado de crença, mas que faz com que inferências se tornem cada vez mais caras à medida que a história cresce.

[13] Observe que a adição de axiomas de precondição significa que não precisamos incluir precondições para ações nos axiomas de estado sucessor.

CAPÍTULO 8

Lógica de primeira ordem

Em que notamos que o mundo é abençoado com muitos objetos, alguns dos quais estão relacionados a outros objetos e em que nos empenhamos para raciocinar sobre eles.

No Capítulo 7, mostramos como um agente baseado em conhecimento poderia representar o mundo em que opera e deduzir que ações executar. Usamos a lógica proposicional como nossa linguagem de representação porque ela foi suficiente para ilustrar os conceitos básicos de lógica e agentes baseados em conhecimento. Infelizmente, a lógica proposicional é uma linguagem muito fraca para representar o conhecimento de ambientes complexos de forma concisa. Neste capítulo, examinaremos a **lógica de primeira ordem**,[1] que é suficientemente expressiva para representar de forma satisfatória nosso conhecimento comum. Ela também compõe ou forma os alicerces de muitas outras linguagens de representação e foi intensivamente estudada por muitas décadas. Começamos na Seção 8.1 com uma descrição das linguagens de representação em geral; a Seção 8.2 focaliza a sintaxe e a semântica da lógica de primeira ordem; as Seções 8.3 e 8.4 ilustram o uso da lógica de primeira ordem em representações simples.

8.1 UMA REVISÃO DA REPRESENTAÇÃO

Nesta seção, discutiremos a natureza das linguagens de representação. Nossa discussão motivará o desenvolvimento da lógica de primeira ordem, uma linguagem muito mais expressiva que a lógica proposicional introduzida no Capítulo 7. Estudaremos a lógica proposicional e outros tipos de linguagens para entender o que funciona e o que não funciona. Nossa discussão será superficial, compactando séculos de pensamento, tentativa e erro em alguns parágrafos.

As linguagens de programação (como C++, Java ou Lisp) são sem dúvida a maior classe de linguagens formais em uso comum. Os programas propriamente ditos representam, em sentido direto, apenas processos computacionais. As estruturas de dados dentro de programas podem representar fatos; por exemplo, um programa poderia usar um array 4 × 4 para representar o conteúdo do mundo de wumpus. Desse modo, a declaração de linguagem de programação *Mundo*[2,2] ← *Poço* é uma forma bastante natural de afirmar que existe um poço no quadrado [2,2]. (Tais representações poderiam ser consideradas *ad hoc*; os sistemas de bancos de dados foram desenvolvidos exatamente

para fornecer um modo mais geral e independente de domínios para armazenar e recuperar fatos.) O que falta às linguagens de programação é algum mecanismo geral para derivar fatos a partir de outros fatos; cada atualização em uma estrutura de dados é feita por um procedimento específico do domínio cujos detalhes são derivados pelo programador a partir de seu próprio conhecimento do domínio. Essa abordagem procedural pode ser comparada com a natureza **declarativa** da lógica proposicional, em que o conhecimento e a inferência estão separados, e a inferência é inteiramente independente de domínios.

Uma segunda desvantagem das estruturas de dados em programas (e também dos bancos de dados) é a falta de qualquer meio fácil de se dizer, por exemplo, que "existe um poço em [2,2] ou [3,1]" ou "Se o wumpus está em [1,1], então ele não está em [2,2]". Os programas podem armazenar um valor único para cada variável, e alguns sistemas permitem que o valor seja "desconhecido", mas lhes falta a capacidade de expressão necessária para manipular informações parciais.

A lógica proposicional é uma linguagem declarativa porque sua semântica se baseia em uma relação-verdade entre sentenças e mundos possíveis. Ela também tem capacidade expressiva suficiente para lidar com informações parciais, usando disjunção e negação. A lógica proposicional tem uma terceira propriedade, que é interessante em linguagens de representação, isto é, a **composicionalidade**. Em uma linguagem composicional, o significado de uma sentença é uma função do significado de suas partes. Por exemplo, "$S_{1,4} \land S_{1,2}$" está relacionado aos significados de "$S_{1,4}$" e "$S_{1,2}$". Seria muito estranho se "$S_{1,4}$" significasse que existe um fedor no quadrado [1,4] e "$S_{1,2}$" significasse que existe um fedor no quadrado [1,2], mas "$S_{1,4} \land S_{1,2}$" significasse que a França e a Polônia empataram por 1×1 na partida de classificação do torneio de futebol da semana passada. É claro que a não omposicionalidade torna muito mais difícil a utilização do sistema de raciocínio.

Entreteanto, como vimos no Capítulo 7, a lógica proposicional se ressente da falta de capacidade de expressão para descrever *de forma concisa* um ambiente com muitos objetos. Por exemplo, fomos forçados a escrever uma regra separada sobre brisas e poços para cada quadrado, como:

$$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1}).$$

Por outro lado, em linguagem natural, parece bastante fácil dizer de uma vez por todas que "quadrados adjacentes a poços têm brisa". A sintaxe e a semântica da linguagem natural tornam possível de algum modo descrever o ambiente de forma concisa.

8.1.1 A linguagem do pensamento

As linguagens naturais (como inglês ou espanhol) na realidade são muito expressivas. Conseguimos escrever este livro quase todo em linguagem natural, apenas com lapsos ocasionais em outras linguagens (inclusive lógica, matemática e a linguagem dos diagramas). Existe uma longa tradição em linguística e na filosofia da linguagem que visualiza a linguagem natural essencialmente como uma linguagem declarativa de representação do conhecimento. Se pudéssemos descobrir as regras para a linguagem natural, poderíamos usá-las nos sistemas de representação e raciocínio e receber o benefício de milhares de bilhões de páginas que foram escritas em linguagem natural.

A visão moderna de linguagem natural é a de que ela serve como um meio de **comunicação** em vez de pura representação. Quando um falante aponta e diz "Olhe!", o ouvinte sabe que, digamos, o Super-homem finalmente apareceu voando sobre os terraços dos prédios, ainda que não pretendêssemos dizer que a sentença "Olhe!" representa esse fato. Em vez disso, o significado da sentença depende tanto da sentença em si quanto do **contexto** em que ela foi dada. É claro que não se poderia armazenar uma sentença como "Olhe!" em uma base de conhecimento e esperar recuperar seu significado sem armazenar também uma representação do contexto — o que traz a questão de como o próprio contexto pode ser representado. As linguagens naturais também sofrem de **ambiguidade**, um problema para uma linguagem de representação. Conforme Pinker (1995) mencionou: "Quando pensam em *bomba*, sem dúvida as pessoas não ficam confusas com o fato de estarem pensando em um artefato explosivo, em um doce ou em um aparelho utilizado para bombear água de uma cisterna — e, se uma palavra pode corresponder a dois ou três pensamentos, então os pensamentos não podem ser palavras."

A famosa **hipótese de Sapir-Whorf** afirma que nossa compreensão do mundo é fortemente influenciada pela língua que falamos. Whorf (1956) escreveu: "Dividimos a natureza, organizamos em conceitos e atribuímos os significados que atribuímos, em grande parte, porque somos partes de um acordo para organizá-la dessa forma — um acordo que mantém toda a nossa comunidade de fala e está codificado nos padrões de nossa língua." É certamente verdade que as diferentes comunidades de fala dividem o mundo de forma diferente. Os franceses têm duas palavras, "*chaise*" e "*fauteuil*", para um conceito para o qual os falantes de inglês têm um: "*chair*" (cadeira). Mas os falantes de inglês conseguem reconhecer facilmente a categoria *fauteuil* e dar-lhe um nome — mais ou menos, "cadeira de braço aberto" — então, a linguagem faz diferença realmente? Whorf baseou-se essencialmente na intuição e especulação, mas nos anos seguintes nós realmente tivemos dados reais de estudos antropológicos, psicológicos e neurológicos.

Por exemplo, você consegue lembrar-se sobre qual das duas sentenças seguintes compôs a abertura da Seção 8.1?

"Nesta seção, discutiremos a natureza das linguagens de representação…"

"Esta seção cobre o tema das linguagens de representação do conhecimento…"

Wanner (1974) fez uma experiência semelhante e constatou que os indivíduos fazem a escolha certa em um nível de chance — cerca de 50% do tempo —, mas lembram-se do conteúdo que leram com mais de 90% de precisão. Isso sugere que as pessoas processam as palavras para formar uma espécie de representação *não verbal*.

Mais interessante é o caso de um conceito que está completamente ausente em um idioma. Os falantes da língua aborígene australiana *guugu yimithirr* não têm palavras para direções relativas, tais como frente, trás, direita ou esquerda. Em vez disso, utilizam direções absolutas, dizendo, por exemplo, o equivalente a "Tenho dor no meu braço do norte". Essa diferença na linguagem traz uma diferença no comportamento: os falantes de *guugu yimithirr* são melhores para navegar em terreno aberto, enquanto os falantes de inglês são melhores em colocar o garfo à direita do prato.

A linguagem também parece influenciar o pensamento através de recursos gramaticais aparentemente arbitrários, como o gênero dos substantivos. Por exemplo, "ponte" é masculino em

espanhol e feminino em alemão. Boroditsky (2003) pediu aos indivíduos para escolher adjetivos ingleses para descrever uma fotografia de uma ponte em particular. Os falantes de espanhol escolheram *grande*, *perigosa*, *forte* e *elevada*; os falantes de alemão escolheram *bonita*, *elegante*, *frágil* e *delgada*. As palavras podem servir como pontos de ancoragem que afetam a maneira como percebemos o mundo. Loftus e Palmer (1974) mostraram um filme de um acidente de carro para indivíduos experimentais. Para o questionamento "A que velocidade iam os carros quando houve o contato?", a resposta foi 51 km por hora, e quando foi usada a palavra "colisão" em vez de "contato" o relato foi uma média de 66 km por hora, para os mesmos carros no mesmo filme.

Em um sistema de raciocínio de lógica de primeira ordem que usa FNC, podemos ver que a forma linguística "¬ (A ∨ B)" e "¬A ∧ ¬ B" é a mesma porque podemos olhar para dentro do sistema e ver que as duas sentenças são armazenadas com a mesma forma canônica FNC. Podemos fazer isso com o cérebro humano? Até recentemente, a resposta era "não", mas agora é "talvez". Mitchell *et al.* (2008) colocaram indivíduos em uma máquina IRMf (imagem por ressonância magnética funcional), mostraram-lhes palavras como "aipo" e formaram a imagem de seus cérebros. Os pesquisadores foram capazes de treinar um programa de computador que previa, a partir da imagem cerebral, que palavra fora apresentada para o indivíduo. Dadas duas opções (por exemplo, "aipo" ou "avião"), o sistema previu corretamente em 77% das vezes. O sistema pode até mesmo prever em um nível superior a 50% palavras que nunca tinha visto em uma imagem IRMf antes (considerando-se as imagens de palavras relacionadas) e de pessoas que nunca havia visto antes (provando que a IRMf revela algum nível de representação comum entre as pessoas). Esse tipo de trabalho ainda está no início, mas a IRMf (e outras tecnologias de imagem comoa eletrofisiologia intracraniana (Sahin *et al.,* 2009)) prometem nos dar ideias muito mais concretas de como são as representações do conhecimento humano.

Do ponto de vista da lógica formal, a representação do mesmo conhecimento de duas maneiras diferentes não faz absolutamente nenhuma diferença; os mesmos fatos serão deriváveis de qualquer representação.

Na prática, porém, uma representação pode exigir menos etapas para obter uma conclusão, o que significa que um raciocinador com recursos limitados poderia chegar à conclusão através de uma representação, mas não através de outra. Para tarefas *não dedutivas*, como aprender da experiência, os resultados dependem *necessariamente* da forma utilizada de representações. Mostraremos no Capítulo 18 que, quando um programa de aprendizado considera duas teorias do mundo possíveis, as duas compatíveis com todos os dados, a forma mais comum de resolver o empate é escolher a teoria mais sucinta — que depende da linguagem utilizada para representar as teorias. Assim, a influência da linguagem sobre o pensamento é inevitável para qualquer agente que faz aprendizado.

8.1.2 Combinando o melhor de linguagens formais e naturais

Podemos adotar os fundamentos da lógica proposicional — uma semântica declarativa, composicional, independente do contexto e não ambígua — e construir uma lógica mais expressiva sobre esses fundamentos, tomando emprestadas ideias de representação da linguagem natural, ao mesmo tempo em que evitamos suas desvantagens. Quando examinamos a sintaxe da linguagem

natural, os elementos mais óbvios são substantivos e sentenças nominais que se referem a **objetos** (quadrados, poços, wumpus), além de verbos e sentenças verbais que se referem a **relações** entre objetos (é arejado, é adjacente a, atira). Algumas dessas relações são **funções** — relações em que existe somente um "valor" para uma dada "entrada". É fácil começar a listar exemplos de objetos, relações e funções:

- Objetos: pessoas, casas, números, teorias, Ronald McDonald, cores, jogos de beisebol, guerras, séculos…
- Relações: podem ser relações unárias ou **propriedades** como vermelho, redondo, falso, primo, de vários pavimentos…, ou relações *n*-árias mais gerais, como irmão de, maior que, interior a, parte de, tem cor, ocorreu depois de, pertence a, fica entre…
- Funções: pai de, melhor amigo, terceiro turno de, uma unidade maior que, início de…

Na verdade, praticamente qualquer asserção pode ser considerada uma referência a objetos e propriedades ou relações. Aqui estão alguns exemplos:

- "Um mais dois é igual a três."

  Objetos: um, dois, três, um mais dois. Relação: é igual a. Função: mais. ("Um mais dois" é um nome para o objeto obtido pela aplicação da função "mais" aos objetos "um" e "dois". Três é outro nome para esse objeto.)
- "Quadrados vizinhos ao wumpus são fedorentos."
  Objetos: wumpus, quadrados. Propriedade: fedorento. Relação: vizinhos.
- "O perverso rei João governou a Inglaterra em 1200."
  Objetos: João, Inglaterra, 1200. Relação: governou. Propriedades: perverso, rei.

A linguagem da **lógica de primeira ordem**, cuja sintaxe e cuja semântica definiremos na próxima seção, é elaborada em torno de objetos e relações. Ela é tão importante para a matemática, a filosofia e a inteligência artificial exatamente porque podemos considerar que esses campos — e, na realidade, grande parte da existência humana diária — refletem o tratamento de objetos e das relações entre eles. A lógica de primeira ordem também pode expressar fatos sobre *alguns* ou *todos* os objetos no universo. Isso nos permite representar leis ou regras gerais, como a declaração: "Quadrados vizinhos ao wumpus são fedorentos."

A principal diferença entre a lógica proposicional e a lógica de primeira ordem reside no **compromisso ontológico** feito por cada linguagem, isto é, o que ela pressupõe sobre a natureza da *realidade*. Matematicamente, esse comprometimento é expresso através da natureza dos **modelos** formais em relação a qual verdade das sentenças foi definida. Por exemplo, a lógica proposicional pressupõe que existem fatos que são válidos ou não são válidos no mundo. Cada fato pode se encontrar em um destes dois estados, verdadeiro ou falso, e cada modelo determina *verdadeiro ou falso* para cada símbolo de proposição (veja a Seção 7.4.2).[2] A lógica de primeira ordem pressupõe mais do que isso; especificamente, que o mundo consiste em objetos com certas relações entre eles que são ou não são válidas. Os modelos formais são correspondentemente mais complicados que os da lógica proposicional. Lógicas com propósitos especiais criam ainda outros compromissos ontológicos; por exemplo, a **lógica temporal** pressupõe que os fatos são válidos em *instantes*

particulares e que esses instantes (que podem ser pontos ou intervalos) estão ordenados. Desse modo, as lógicas com propósitos especiais dão a certos tipos de objetos (e aos axiomas sobre eles) *status* de “primeira classe” dentro da lógica, em vez de simplesmente definirem esses objetos na base de conhecimento. A **lógica de alta ordem** visualiza as relações e funções referidas à lógica de primeira ordem como objetos em si. Isso permite que se façam asserções sobre *todas* as relações — por exemplo, poderíamos desejar definir o que significa o fato de uma relação ser transitiva. Diferentemente da maioria das lógicas com propósitos especiais, a lógica de alta ordem é estritamente mais expressiva que a lógica de primeira ordem, no sentido de que algumas sentenças de lógica de alta ordem não podem ser expressas por qualquer número finito de sentenças de lógica de primeira ordem.

Uma lógica também pode ser caracterizada por seus **compromissos epistemológicos** — os estados possíveis de conhecimento que ela permite a respeito de cada fato. Tanto na lógica proposicional quanto na lógica de primeira ordem, uma sentença representa um fato, e o agente acredita que a sentença é verdadeira, acredita que ela é falsa ou não tem nenhuma opinião. Então, essas lógicas têm três estados possíveis de conhecimento a respeito de qualquer sentença. Por outro lado, os sistemas que utilizam a **teoria da probabilidade** podem ter qualquer *grau de crença*, variando de 0 (descrença total) até 1 (crença total).[3] Por exemplo, um agente probabilístico de mundo de wumpus poderia acreditar que o wumpus está em [1,3] com probabilidade 0,75. Os compromissos ontológicos e epistemológicos de cinco lógicas diferentes estão resumidos na Figura 8.1.

Linguagem	Compromisso ontológico (o que existe no mundo)	Compromisso epistemológico (a crença de um agente sobre os fatos)
Lógica proposicional	fatos	verdadeiro/falso/desconhecido
Lógica de primeira ordem	fatos, objetos, relações	verdadeiro/falso/desconhecido
Lógica temporal	fatos, objetos, relações, tempo	verdadeiro/falso/desconhecido
Teoria da probabilidade	fatos	grau de crença $\in [0, 1]$
Lógica difusa	fatos com grau de verdade $\in [0, 1]$	valor de intervalo conhecido

Figura 8.1 Linguagens formais e seus compromissos ontológicos e epistemológicos.

Na próxima seção, iniciaremos o estudo dos detalhes da lógica de primeira ordem. Da mesma maneira que um aluno de física deve ter alguma familiaridade com matemática, um aluno de IA deve desenvolver um talento para trabalhar com notação lógica. Por outro lado, também é importante *não* ficar muito preocupado com as *particularidades* da notação lógica — afinal, existem dezenas de versões diferentes. Os principais itens a apreender são a forma como a linguagem facilita representações concisas e como sua semântica leva a procedimentos de raciocínio consistentes.

8.2 SINTAXE E SEMÂNTICA DA LÓGICA DE PRIMEIRA ORDEM

Iniciaremos esta seção especificando com maior exatidão a ideia de que os mundos possíveis da lógica de primeira ordem refletem o compromisso ontológico com objetos e relações. Em seguida, introduziremos os vários elementos da linguagem, explicando sua semântica à medida que prosseguirmos.

8.2.1 Modelos para lógica de primeira ordem

Vimos, no Capítulo 7, que os modelos de uma linguagem lógica são as estruturas formais que constituem os mundos possíveis sob consideração. Cada modelo liga o vocabulário das sentenças lógicas aos elementos do mundo possível, para que a verdade de qualquer sentença possa ser determinada. Assim, modelos de lógica proposicional ligam símbolos de proposição com valores-verdade predefinidos. Os modelos para lógica de primeira ordem são muito mais interessantes. Em primeiro lugar, eles contêm objetos! O **domínio** de um modelo é o conjunto de objetos ou **elementos do domínio** que ele contém. Exige-se que o domínio seja *não vazio* — todos os mundos possíveis devem conter pelo menos um objeto (veja o Exercício 8.7 para uma discussão de mundos vazios). Matematicamente falando, não importa o *que* são esses objetos — tudo o que importa é *quantos* há em cada modelo em particular —, mas, para fins pedagógicos, vamos utilizar um exemplo concreto. A Figura 8.2 mostra um modelo com cinco objetos: Ricardo Coração de Leão, rei da Inglaterra de 1189 até 1199; seu irmão mais jovem, o perverso rei João, que governou de 1199 até 1215; a perna esquerda de Ricardo e de João; uma coroa.

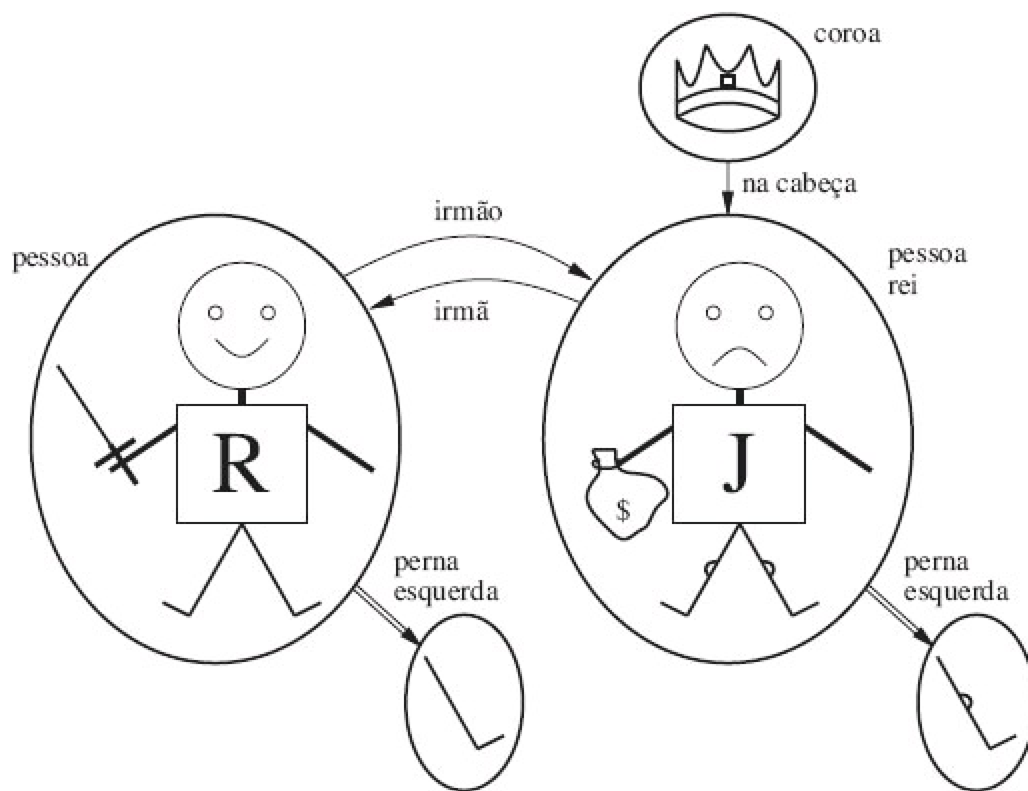


Figura 8.2 Modelo contendo cinco objetos, duas relações binárias, três relações unárias (indicadas por rótulos nos objetos) e uma função unária, perna esquerda.

Os objetos no modelo podem estar *relacionados* de diversas maneiras. Na figura, Ricardo e João são irmãos. Formalmente falando, uma relação é apenas o conjunto de **tuplas** de objetos inter-

relacionados. (Uma tupla é uma coleção de objetos organizados em uma ordem fixa e é representada por colchetes angulares em torno dos objetos.) Desse modo, a relação "irmão" nesse modelo é o conjunto

$$\{\langle \text{Ricardo Coração de Leão, Rei João} \rangle, \langle \text{Rei João, Ricardo Coração de Leão} \rangle\}. \qquad (8.1)$$

(Aqui nos referimos aos objetos em linguagem natural, mas você pode, se desejar, substituir mentalmente os nomes pelas figuras.) A coroa está na cabeça do rei João, então a relação "na cabeça" contém apenas uma tupla, ⟨coroa, rei João⟩. As relações "irmão" e "na cabeça" são relações binárias, isto é, relacionam pares de objetos. O modelo também contém relações unárias ou propriedades: a propriedade "pessoa" é verdadeira para Ricardo e para João; a propriedade "rei" é verdadeira apenas para João (presumivelmente porque nesse momento Ricardo está morto); e a propriedade "coroa" é verdadeira apenas para coroa.

É melhor pensar em certos tipos de relacionamentos como se fossem funções, nas quais determinado objeto deve estar relacionado a exatamente um objeto desse modo. Por exemplo, cada pessoa tem uma perna esquerda, então o modelo tem uma função unária "perna esquerda" que inclui os seguintes mapeamentos:

$$\begin{aligned}&\langle \text{Ricardo Coração de Leão} \rangle \rightarrow \text{perna esquerda de Ricardo}\\&\langle \text{Rei João} \rangle \rightarrow \text{perna esquerda de João.}\end{aligned} \qquad (8.2)$$

Estritamente falando, modelos em lógica de primeira ordem exigem **funções totais**, isto é, deve haver um valor para toda tupla de entrada. Desse modo, a coroa deve ter uma perna esquerda assim como cada uma das pernas esquerdas deve ter uma perna esquerda. Existe uma solução técnica para esse problema esquisito envolvendo um objeto "invisível" adicional que é a perna esquerda de tudo que não tem perna esquerda, inclusive ele próprio. Felizmente, desde que não se faça nenhuma afirmação sobre as pernas esquerdas de itens que não têm pernas esquerdas, esses detalhes técnicos não têm nenhuma importância.

Até agora, descrevemos os elementos que preenchem modelos de lógica de primeira ordem. Outra parte essencial de um modelo é a ligação entre esses elementos e o vocabulário das sentenças lógicas, que explicaremos a seguir.

8.2.2 Símbolos e interpretações

Vejamos agora a sintaxe da linguagem. O leitor impaciente pode obter uma descrição completa da gramática formal da lógica de primeira ordem na Figura 8.3.

Sentença	→	*SentençaAtômica* | *SentençaComplexa*
SentençaAtômica	→	*PredicadoPredicado*(*Termo*, . . .) | *Termo* = *Termo*
SentençaComplexa	→	(*Sentença*) | [*Sentença*]
	|	¬ *Sentença*
	|	*Sentença* ∧ *Sentença*
	|	*Sentença* ∨ *Sentença*
	|	*Sentença* ⇒ *Sentença*
	|	*Sentença* ⇔ *Sentença*
	|	*Quantificador*, *Variável*.... *Sentença*
Termo	→	*Função*(*Termo*, . . .)
	|	*Constante*
	|	*Variável*
Quantificador	→	∀ | ∃
Constante	→	*A* | *X*1 | *João* | . . .
Variável	→	*a* | *x* | *s* | . . .
Predicado	→	*Verdadeiro* |*Falso*|*Depois*|*Ama*| *Chovendo* | . . .
Função	→	*Mãe* | *PernaEsquerda*| . . .
Precedência de operador	:	¬, =, ∧, ∨, ⇒, ⇔

Figura 8.3 Sintaxe da lógica de primeira ordem com igualdade, especificada na forma de Backus-Naur. A precedência dos operadores é especificada do mais alto para o mais baixo. A precedência dos quantificadores é tal que um quantificador se aplica a tudo à sua direita.

Os elementos sintáticos básicos da lógica de primeira ordem são os símbolos que representam objetos, relações e funções. Por essa razão, os símbolos são de três tipos: **símbolos de constantes**, que representam objetos, **símbolos de predicados**, que representam relações, e **símbolos de funções**, que representam funções. Adotaremos a convenção de que esses símbolos começarão com letras maiúsculas. Por exemplo, poderíamos usar os símbolos de constantes *Ricardo* e *João*, os símbolos de predicados, *Irmão*, *NaCabeça*, *Pessoa*, *Rei* e *Coroa* e o símbolo de função *PernaEsquerda*. Como ocorre com os símbolos de proposições, a escolha de nomes cabe inteiramente ao usuário. Cada símbolo de predicado e de função vem com uma **aridade** que fixa o número de argumentos.

Como na lógica proposicional, cada modelo deve fornecer a informação necessária para determinar se dada sentença é verdadeira ou falsa. Assim, além de seus objetos, relações e funções, cada modelo inclui uma **interpretação** que especifique exatamente quais objetos, relações e funções são referidos pelos símbolos de constantes, predicados e funções. Uma interpretação possível para nosso exemplo — que um lógico chamaria de **interpretação pretendida** — é:

- *Ricardo* se refere a Ricardo Coração de Leão e *João* se refere ao perverso rei João.
- *Irmão* se refere à relação de fraternidade, ou seja, o conjunto de tuplas de objetos dado na Equação 8.1; *NaCabeça* se refere à relação "na cabeça" que é válida entre a coroa e o rei João; *Pessoa*, *Rei* e *Coroa* se referem aos conjuntos de objetos que são pessoas, reis e coroas.

- *PernaEsquerda* se refere *à função "perna esq*uerda", isto é, ao mapeamento definido pela Equação 8.2.

Certamente existem muitas outras interpretações possíveis. Por exemplo, uma interpretação mapeia *Ricardo* à coroa e *João* à perna esquerda do rei João. Existem cinco objetos no modelo; portanto, existem 25 interpretações possíveis apenas para os símbolos de constantes *Ricardo* e *João*. Note que nem todos os objetos precisam ter um nome — por exemplo, a interpretação pretendida não atribui nomes à coroa ou às pernas. Também é possível um objeto ter vários nomes; existe uma interpretação sob a qual tanto *Ricardo* quanto *João* se referem à coroa.[4] Se achar essa possibilidade confusa, lembre-se de que, em lógica proposicional, é perfeitamente possível haver um modelo em que *Nublado* e *Ensolarado* sejam verdadeiros; cabe à base de conhecimento eliminar modelos inconsistentes com nosso conhecimento.

Em resumo, um modelo em lógica de primeira ordem consiste em um conjunto de objetos e uma interpretação que mapeia de símbolos de constantes a objetos, de símbolos de predicados às relações sobre esses objetos, e de símbolos de função às funções desses objetos. Assim como com a lógica proposicional, a consequência lógica, a validade, e assim por diante, são definidas em termos de *todos os modelos possíveis*. Para se ter uma ideia de como se parece o conjunto de todos os modelos possíveis, veja a Figura 8.4. Ela mostra que os modelos variam dependendo de quantos objetos contêm — de um até o infinito — e na forma como os símbolos de constantes mapeiam os objetos. Se houver dois símbolos de constantes e um objeto, ambos os símbolos devem se referir ao mesmo objeto, mas isso ainda pode acontecer até mesmo com mais objetos. Quando houver mais objetos que símbolos de constantes, alguns dos objetos não terão nomes. Devido ao número de modelos possíveis ser ilimitado, verificar a consequência lógica pela enumeração de todos os modelos possíveis não é viável para a lógica de primeira ordem (ao contrário da lógica proposicional). Mesmo se o número de objetos for restrito, o número de combinações pode ser muito grande (veja o Exercício 8.5). Por exemplo, na Figura 8.4, existem 137.506.194.466 modelos com seis ou menos objetos.

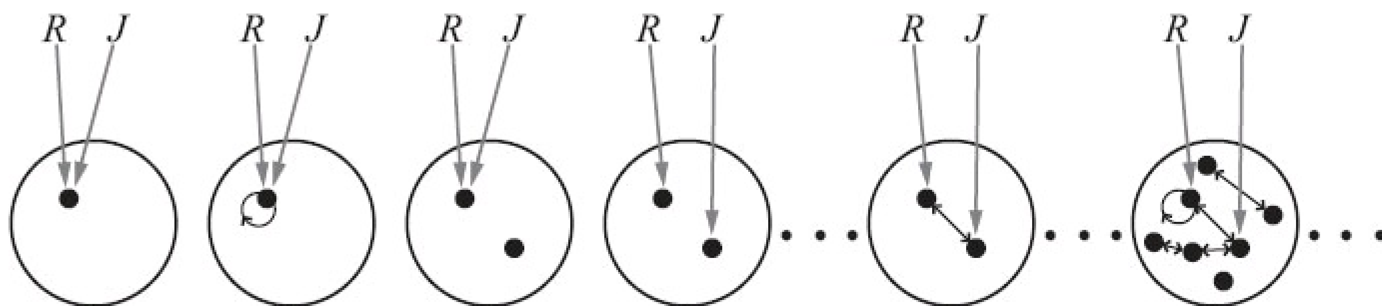


Figure 8.4 Alguns membros do conjunto de todos os modelos para uma linguagem com dois símbolos de constantes, *R e J*, e um símbolo de relação binária. A interpretação de cada símbolo de constante é mostrada pela seta cinza. Dentro de cada modelo, os objetos relacionados são conectados por setas.

8.2.3 Termos

Um **termo** é uma expressão lógica que se refere a um objeto. Os símbolos de constantes são

portanto termos, mas nem sempre é conveniente ter um símbolo distinto para identificar todo objeto. Por exemplo, em linguagem natural poderíamos usar a expressão "perna esquerda do rei João" em lugar de dar um nome à perna. Essa é a finalidade dos símbolos de funções: em vez de usar um símbolo de constante, utilizamos *PernaEsquerda*(*João*). No caso geral, um termo complexo é formado por um símbolo de função seguido por uma lista entre parênteses de termos como argumentos para o símbolo de função. É importante lembrar que um termo complexo é apenas uma espécie complicada de nome. Não é uma "chamada de sub-rotina" que "retorna um valor". Não existe nenhuma sub-rotina *PernaEsquerda* que receba uma pessoa como entrada e retorne uma perna.

Podemos raciocinar sobre pernas esquerdas (por exemplo, enunciando a regra geral de que todo mundo tem uma, e depois deduzindo que João deve ter uma perna esquerda), sem sequer fornecermos uma definição de *PernaEsquerda*. Isso é algo que não pode ser feito com sub-rotinas em linguagens de programação.[5]

A semântica formal dos termos é direta. Considere um termo $f(t_1,\ldots, t_n)$. O símbolo de função f se refere a alguma função no modelo (vamos chamá-la F); os termos de argumentos se referem a objetos no domínio (vamos chamá-los $d_1,\ldots, d_n$); e o termo como um todo se refere ao objeto que é o valor da função F aplicada a $d_1,\ldots, d_n$. Por exemplo, suponha que o símbolo de função *PernaEsquerda* se refira à função mostrada na Equação 8.2 e *João* se refira ao rei João; então, *PernaEsquerda*(*João*) se refere à perna esquerda do rei João. Desse modo, a interpretação fixa o referente de todo termo.

8.2.4 Sentenças atômicas

Agora que temos termos para fazer referência a objetos e símbolos de predicados para fazer referência a relações, podemos reuni-los para formar **sentenças atômicas** que enunciam fatos.

Uma **sentença atômica** (ou **átomo**, para encurtar) é formada a partir de um símbolo de predicado, seguido por uma lista de termos entre parênteses:

Irmão(*Ricardo*, *João*)

Isso enuncia, sob a interpretação pretendida apresentada anteriormente, que Ricardo Coração de Leão é o irmão do rei João.[6] As sentenças atômicas podem ter termos complexos como argumentos. Desse modo,

Casado(*Pai*(*Ricardo*), *Mãe*(*João*))

enuncia que o pai de Ricardo Coração de Leão é casado com a mãe do rei João (mais uma vez, sob uma interpretação apropriada).

☞ *Uma sentença atômica é* ***verdadeira*** *em dado modelo, sob dada interpretação, se a relação referida pelo símbolo de predicado é válida entre os objetos referidos pelos argumentos.*

8.2.5 Sentenças complexas

Podemos usar **conectivos lógicos** para construir sentenças mais complexas, com a mesma sintaxe e semântica que no cálculo proposicional. Aqui estão quatro sentenças que são verdadeiras no modelo da Figura 8.2, sob nossa interpretação pretendida:

$\neg Irmão(PernaEsquerda(Ricardo), João)$
$Irmão(Ricardo, João) \land Irmão(João, Ricardo)$
$Rei(Ricardo) \lor Rei(João)$
$\neg Rei(Ricardo) \Rightarrow Rei(João)$

8.2.6 Quantificadores

Uma vez que temos uma lógica que permite objetos, não deixa de ser natural querer expressar propriedades de coleções inteiras de objetos, em vez de enumerar os objetos pelo nome. Os **quantificadores** nos permitem fazê-lo. A lógica de primeira ordem contém dois quantificadores-padrão, chamados *universal* e *existencial*.

Quantificação universal (∀)

Lembre-se da dificuldade que tivemos no Capítulo 7 com a expressão de regras gerais em lógica proposicional. Regras como "Quadrados vizinhos ao wumpus são fedorentos" e "Todos os reis são pessoas" são comuns em lógica de primeira ordem. Lidaremos com a primeira dessas regras na Seção 8.3. A segunda regra, "Todos os reis são pessoas", é escrita em lógica de primeira ordem como

$\forall x\ Rei(x) \Rightarrow Pessoa(x)$.

Normalmente, ∀ é lido como "Para todo…" ou "Para todos…" (lembre-se de que o A invertido representa "todos" ou "*all*" em inglês). Desse modo, a sentença afirma que "Para todo x, se x é um rei, então x é uma pessoa".

O símbolo x é chamado **variável**. Por convenção, as variáveis são letras minúsculas. Uma variável é um termo por si só e, como tal, também pode servir como o argumento de uma função — por exemplo, $PernaEsquerda(x)$. Um termo sem variáveis é chamado **termo base** (ground term).

Intuitivamente, a sentença $\forall x\ P$, onde P é qualquer expressão lógica, afirma que P é verdadeira para todo objeto x. Mais precisamente, $\forall x\ P$ é verdadeira em dado modelo se P é verdadeira em todas as **interpretações estendidas** possíveis construídas a partir da interpretação dada ao modelo, em que cada interpretação estendida especifica um elemento de domínio ao qual x se refere.

Isso parece complicado, mas, na realidade, é apenas um modo cuidadoso de declarar o significado intuitivo da quantificação universal. Considere o modelo mostrado na Figura 8.2 e a interpretação pretendida que o acompanha. Podemos estender a interpretação de cinco maneiras:

$x \rightarrow$ Ricardo Coração de Leão,
$x \rightarrow$ rei João,
$x \rightarrow$ perna esquerda de Ricardo,
$x \rightarrow$ perna esquerda de João,
$x \rightarrow$ a coroa.

A sentença universalmente quantificada $\forall x\ Rei(x) \Rightarrow Pessoa(x)$ é verdadeira no modelo original se a sentença $Rei(x) \Rightarrow Pessoa(x)$ é verdadeira sob cada uma das cinco interpretações estendidas. Ou seja, a sentença universalmente quantificada é equivalente a afirmar as cinco sentenças a seguir:

Ricardo Coração de Leão é um rei ⇒ Ricardo Coração de Leão é uma pessoa.
O rei João é um rei ⇒ O rei João é uma pessoa.
A perna esquerda de Ricardo é um rei ⇒ A perna esquerda de Ricardo é uma pessoa.
A perna esquerda de João é um rei ⇒ A perna esquerda de João é uma pessoa.
A coroa é um rei ⇒ A coroa é uma pessoa.

Vamos examinar cuidadosamente esse conjunto de asserções. Tendo em vista que em nosso modelo o rei João é o único rei, a segunda sentença afirma que ele é uma pessoa, como seria de esperar. Porém, e no caso das outras quatro sentenças, que parecem fazer afirmações sobre pernas e coroas? Isso faz parte do significado de "Todos os reis são pessoas"? De fato, as outras quatro asserções são verdadeiras no modelo, mas não fazem nenhuma afirmação sobre as qualificações pessoais de pernas, coroas ou nem mesmo de Ricardo. Isso ocorre porque nenhum desses objetos é um rei. Observando a tabela-verdade para ⇒ (Figura 7.8), vemos que a implicação é verdadeira sempre que sua premissa é falsa, *não importando* o valor-verdade da conclusão. Desse modo, afirmando a sentença universalmente quantificada, equivalente a afirmar uma lista inteira de implicações individuais, acabamos afirmando a conclusão da regra apenas para os objetos para os quais a premissa é verdadeira, e não dizendo absolutamente nada sobre os indivíduos para os quais a premissa é falsa. Portanto, a definição de ⇒ da tabela-verdade se mostra perfeita para a escrita de regras gerais com quantificadores universais.

Um equívoco comum, que ocorre com frequência, mesmo no caso de leitores diligentes que leram esse parágrafo várias vezes, é usar a conjunção em vez da implicação. A sentença

$\forall x\ Rei(x) \land Pessoa(x)$

seria equivalente a afirmar

Ricardo Coração de Leão é um rei ∧ Ricardo Coração de Leão é uma pessoa,
Rei João é um rei ∧ Rei João é uma pessoa,
A perna esquerda de Ricardo é um rei ∧ A perna esquerda de Ricardo é uma pessoa,

e assim por diante. É óbvio que isso não capta o que queremos.

Quantificação existencial (∃)

A quantificação universal faz declarações sobre todo objeto. De modo semelhante, podemos fazer

uma declaração sobre *algum* objeto no universo sem nomeá-lo, utilizando um quantificador existencial. Por exemplo, para dizer que o rei João tem uma coroa em sua cabeça, escrevemos:

$$\exists x\ Coroa(x) \land NaCabeça(x, João).$$

$\exists x$ é lido como "Existe um x tal que..." ou "Para algum x...".

Intuitivamente, a sentença $\exists x\ P$ afirma que P é verdadeira para pelo menos um objeto x. Mais precisamente, $\exists x\ P$ é verdadeira em dado modelo sob dada interpretação se P é verdadeira em *pelo menos uma* interpretação estendida que atribua x a um elemento de domínio. Ou seja, pelo menos uma das afirmações a seguir deve ser verdadeira:

Ricardo Coração de Leão é uma coroa ∧ Ricardo Coração de Leão está na cabeça de João;
Rei João é uma coroa ∧ Rei João está na cabeça de João;
A perna esquerda de Ricardo é uma coroa ∧ A perna esquerda de Ricardo está na cabeça de João;
A perna esquerda de João é uma coroa ∧ A perna esquerda de João está na cabeça de João;
A coroa é uma coroa ∧ A coroa está na cabeça de João.

A quinta afirmação é verdadeira no modelo e, assim, a sentença existencialmente quantificada original é verdadeira no modelo. Note que, por nossa definição, a sentença também seria verdadeira em um modelo no qual o rei João estivesse usando duas coroas. Isso é inteiramente consistente com a sentença original "O rei João tem uma coroa em sua cabeça".[7]

Da mesma maneira que ⇒ parece ser o conectivo natural a usar com ∀, ∧ é o conectivo natural a usar com ∃. O uso de ∧ como o principal conectivo com ∀ levou a uma declaração forte demais no exemplo da seção anterior; o uso de ⇒ com ∃ em geral conduz a uma declaração muito fraca. Considere a sentença a seguir:

$$\exists x\ Coroa(x) \Rightarrow NaCabeça(x, João).$$

À primeira vista, talvez ela pareça ser uma apresentação razoável de nossa sentença. Aplicando a semântica, vemos que a sentença afirma que pelo menos uma das asserções a seguir é verdadeira:

Ricardo Coração de Leão é uma coroa ⇒ Ricardo Coração de Leão está na cabeça de João;
Rei João é uma coroa ⇒ Rei João está na cabeça de João;
A perna esquerda de Ricardo é uma coroa ⇒ A perna esquerda de Ricardo está na cabeça de João;

e assim por diante. Agora uma implicação é verdadeira se tanto a premissa quanto a conclusão são verdadeiras *ou se sua premissa é falsa*. Assim, se Ricardo Coração de Leão não é uma coroa, a primeira afirmação é verdadeira e a existencial é satisfeita. Desse modo, uma sentença de implicação existencialmente quantificada é verdadeira quando *qualquer* objeto falhar em satisfazer a premissa; por conseguinte, tais sentenças de fato não dizem muito.

Quantificadores aninhados

Com frequência, desejaremos expressar sentenças mais complexas usando vários quantificadores. O caso mais simples é aquele em que os quantificadores são do mesmo tipo. Por exemplo, a sentença "Irmãos são parentes" pode ser escrita como:

$$\forall x \ \forall y \ Irmão(x, y) \Rightarrow Parente(x, y).$$

Quantificadores consecutivos do mesmo tipo podem ser escritos como um único quantificador com diversas variáveis. Por exemplo, para dizer que o parentesco é um relacionamento simétrico, podemos escrever:

$$\forall x, y \ Parente(x, y) \Leftrightarrow Parente(y, x).$$

Em outros casos, teremos misturas. "Todo mundo ama alguém" quer dizer que, para toda pessoa, existe alguém que essa pessoa ama:

$$\forall x \ \exists y \ Ama(x, y).$$

Por outro lado, para dizer que "Existe alguém que é amado por todo mundo", escrevemos:

$$\exists y \ \forall x \ Ama(x, y).$$

Portanto, a ordem de quantificação é muito importante. Ela fica mais clara se inserimos parênteses. $\forall x \ (\exists y \ Ama(x, y))$ nos diz que *todo mundo* tem uma propriedade específica, ou seja, a propriedade de que alguém os ama. Por outro lado, $\exists x \ (\forall y \ Ama(x, y))$ nos informa que *alguém* no mundo tem uma propriedade específica, isto é, a propriedade de ser amado por todo mundo.

É possível que surja alguma confusão quando dois quantificadores forem usados com o mesmo nome de variável. Considere a sentença

$$\forall x \ (Coroa(x) \lor (\exists x \ Irmão(Ricardo, x))).$$

Nesse caso, o x em $Irmão(Ricardo, x)$ é *existencialmente* quantificado. De acordo com a regra, a variável pertence ao quantificador mais interno que a menciona; então, ela não estará sujeita a qualquer outra quantificação. Outro modo de pensar sobre isso é $\exists x \ Irmão(Ricardo, x)$ é uma sentença sobre Ricardo (afirmando que ele tem um irmão) e não sobre x; assim, a colocação de $\forall x$ do lado de fora dos parênteses não tem nenhum efeito.

Isso poderia ser escrito igualmente bem como $\exists z \ Irmão(Ricardo, z)$. Tendo em vista que isso pode ser a origem de alguma confusão, sempre utilizaremos nomes de variáveis diferentes com quantificadores aninhados.

Conexões entre ∀ e ∃

Na realidade, os dois quantificadores estão intimamente conectados um ao outro por meio da negação. Afirmar que todo mundo detesta cenouras é o mesmo que afirmar que não existe alguém que goste delas e vice-versa:

$\forall x \neg Gosta(x, Cenouras)$ é equivalente a $\neg \exists x\ Gosta(x, Cenouras)$.

Podemos ir um passo adiante: “Todo mundo gosta de sorvete” significa que não existe ninguém que não goste de sorvete:

$\forall x\ Gosta(x, Sorvete)$ é equivalente a $\neg \exists x \neg Gosta(x, Sorvete)$.

Tendo em vista que $\forall$ é realmente uma conjunção sobre o universo de objetos e $\exists$ é uma disjunção, não deve surpreender que eles obedeçam às regras de De Morgan. As regras de De Morgan para sentenças quantificadas e não quantificadas são as seguintes:

$$\begin{array}{llll} \forall x \neg P \equiv \neg \exists x\ P & & \neg(P \vee Q) \equiv \neg P \wedge \neg Q \\ \neg \forall x\ P \equiv \exists x \neg P & & \neg(P \wedge Q) \equiv \neg P \vee \neg Q \\ \forall x\ P \equiv \neg \exists x \neg P & & P \wedge Q \equiv \neg(\neg P \vee \neg Q) \\ \exists x\ P \equiv \neg \forall x \neg P & & P \vee Q \equiv \neg(\neg P \wedge \neg Q). \end{array}$$

Desse modo, na realidade, não precisamos de ambos $\forall$ e de $\exists$, da mesma forma que não precisamos realmente de $\wedge$ e $\vee$. Ainda assim, a legibilidade é mais importante que a parcimônia e, portanto, continuaremos utilizando ambos os quantificadores.

8.2.7 Igualdade

A lógica de primeira ordem inclui mais um meio de criar sentenças atômicas, além de usar um predicado e termos da maneira descrita anteriormente. Podemos usar o **símbolo de igualdade** significando que dois termos se referem ao mesmo objeto. Por exemplo,

$Pai(João) = Henrique$

nos diz que o objeto referido por *Pai*(*João*) e o objeto referido por *Henrique* são iguais. Como uma interpretação fixa o referente de qualquer termo, a determinação da verdade de uma sentença de igualdade é simplesmente uma questão de ver que os referentes dos dois termos são o mesmo objeto.

O símbolo de igualdade pode ser usado para enunciar fatos sobre uma dada função, como acabamos de fazer para o símbolo *Pai*. Ele também pode ser empregado com a negação para insistir no fato de que dois termos não são o mesmo objeto. Para dizer que Ricardo tem pelo menos dois irmãos, escreveríamos:

$\exists x, y\ Irmão(x, Ricardo) \wedge Irmão(y, Ricardo) \wedge \neg(x = y)$.

A sentença

$\exists x, y\ Irmão(x, Ricardo) \wedge Irmão(y, Ricardo)$

não tem o significado pretendido. Em particular, ela é verdadeira no modelo da Figura 8.2, em que Ricardo tem apenas um irmão. Para constatar esse fato, considere a interpretação estendida em que tanto x quanto y são atribuídos ao rei João. A adição de $\neg(x = y)$ elimina tais modelos. A notação $x \neq$

y às vezes é usada como abreviatura para $\neg(x = y)$.

8.2.8 Uma semântica alternativa?

Continuando o exemplo da seção anterior, suponha que acreditemos que Ricardo tem dois irmãos, João e Godofredo.[8] Podemos capturar esse estado de coisas, afirmando

$$Irmão\ (João\ Ricardo) \wedge Irmão\ (Godofredo,\ Ricardo)? \quad (8.3)$$

Não é bem assim. Em primeiro lugar, essa afirmação é verdadeira em um modelo em que Ricardo tem apenas um irmão — precisamos adicionar $John \neq Godofredo$. Segundo, a sentença não descarta modelos em que Ricardo tenha muito mais irmãos além de Godofrredo e João. Assim, a tradução correta de "os irmãos de Ricardo são João e Godofredo" é a seguinte:

$$Irmão(João,\ Ricardo) \wedge Irmão(Godofredo,\ Ricardo) \wedge João \neq Godofredo$$
$$\wedge \forall x\ Irmão\ (x,\ Ricardo) \Rightarrow (x = João \vee x = Godofredo).$$

Para muitos propósitos, isso parece muito mais enfadonho do que a expressão em linguagem natural correspondente. Como consequência, os seres humanos podem cometer erros na tradução do seu conhecimento em lógica de primeira ordem, resultando em comportamentos não intuitivos de sistemas de raciocínio lógico que usam o conhecimento. Podemos conceber uma semântica que permita uma expressão lógica mais simples?

Uma proposta que é muito popular em sistemas de banco de dados funciona da seguinte forma. Em primeiro lugar, insistimos que cada símbolo constante refere-se a um objeto distinto — a chamada **hipótese de nomes únicos**. Em segundo lugar, assumimos que as sentenças atômicas que não sabemos se são verdadeiras são na verdade falsas — a **hipótese de mundo fechado**. Finalmente, invocamos o **fechamento de domínio**, o que significa que cada modelo não contém mais elementos de domínio do que os nomeados pelos símbolos constantes. Sob a semântica resultante, que chamamos de **semântica de banco de dados** para distingui-la da semântica-padrão da lógica de primeira ordem, a equação da sentença 8.3 de fato afirma que os dois irmãos de Ricardo são João e Godofredo. A semântica de banco de dados é também usada em sistemas de lógica de programação, como explicado na Seção 9.4.5.

É instrutivo considerar o conjunto de todos os modelos possíveis sob a semântica de banco de dados para o mesmo caso, como mostrado na Figura 8.4. A Figura 8.5 mostra alguns dos modelos, que vai desde o modelo sem tuplas que satisfaçam a relação até o modelo com todas as tuplas que satisfaçam a relação. Com dois objetos, existem quatro tuplas possíveis de dois elementos, por isso há $2^4 = 16$ diferentes subconjuntos de tuplas que podem satisfazer a relação. Assim, existem 16 modelos possíveis ao todo — bem menos que a infinita quantidade de modelos para a semântica padrão de primeira ordem. Por outro lado, a semântica de banco de dados requer conhecimento preciso do que o mundo contém.

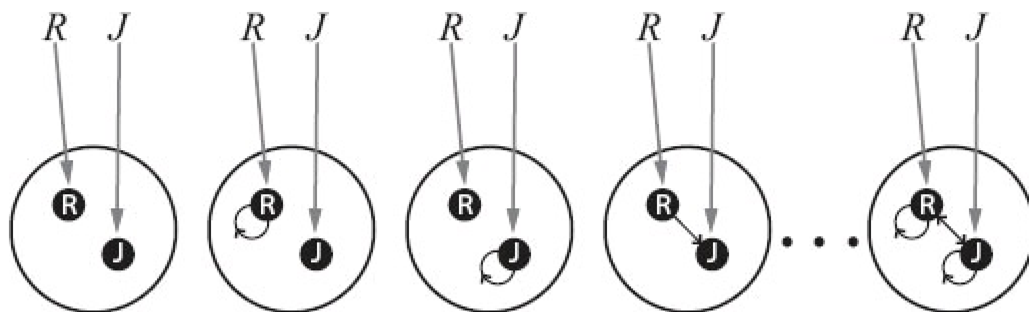


Figura 8.5 Alguns membros do conjunto de todos os modelos de uma linguagem com dois símbolos de constantes, *R* e *J*, e um símbolo de relação binária, sob a semântica de banco de dados. A interpretação dos símbolos de constantes é fixa e há um objeto distinto para cada símbolo de constante.

Esse exemplo traz à tona um ponto importante: não há uma semântica "correta" para a lógica. A utilidade de qualquer semântica proposta depende de quão concisa e intuitiva ela torna a expressão dos tipos de conhecimento que queremos descrever e o quanto é fácil e natural desenvolver as respectivas regras de inferência. A semântica de banco de dados é mais útil quando temos certeza sobre a identidade de todos os objetos descritos na base de conhecimento e quando temos todos os fatos à mão; em outros casos, é quase impraticável. Para o resto deste capítulo, assumiremos a semântica-padrão, enquanto observamos casos em que essa escolha leva a expressões desajeitadas.

8.3 UTILIZAÇÃO DA LÓGICA DE PRIMEIRA ORDEM

Agora que definimos uma linguagem lógica expressiva, é hora de aprender a usá-la. A melhor maneira de fazer isso é empregar exemplos. Vimos algumas sentenças simples que ilustram os diversos aspectos da sintaxe lógica; nesta seção, forneceremos representações mais sistemáticas de alguns **domínios** simples. Em representação do conhecimento, um domínio é simplesmente alguma parte do mundo sobre a qual desejamos expressar algum conhecimento.

Começaremos com uma breve descrição da interface TELL/ASK para bases de conhecimento de primeira ordem. Em seguida, examinaremos os domínios de relacionamentos de família, números, conjuntos e listas, e também o mundo de wumpus. A próxima seção contém um exemplo mais significativo (circuitos eletrônicos), e o Capítulo 12 aborda tudo o que existe no universo.

8.3.1 Asserções e consultas em lógica de primeira ordem

As sentenças são adicionadas a uma base de conhecimento usando-se TELL, exatamente como na lógica proposicional. Tais sentenças são chamadas **asserções**. Por exemplo, podemos afirmar que João é um rei, Ricardo é uma pessoa e que reis são pessoas:

TELL(*BC*, *Rei*(*João*)).
TELL(*BC*, Pessoa(*Ricardo*)).
TELL(*BC*, $\forall$ *x Rei*(*x*) $\Rightarrow$ *Pessoa*(*x*)).

Podemos formular perguntas sobre a base de conhecimento utilizando ASK. Por exemplo,

ASK(*BC*, *Rei*(*João*))

retorna *verdadeiro*. Perguntas formuladas com o uso de ASK são chamadas **consultas** ou **metas**. De modo geral, qualquer consulta que seja consequência lógica da base de conhecimento deve ser respondida afirmativamente. Por exemplo, dadas as duas asserções precedentes, a consulta

ASK(*BC*, *Pessoa*(*João*))

também deve retornar *verdadeiro*. Além disso, podemos formular consultas quantificadas, como:

ASK(*BC*, $\exists$ *x Pessoa*(*x*)).

A resposta é *verdadeira*, mas isso talvez não seja tão útil como gostaríamos. Isso é bem semelhante a responder à pergunta "Pode me dizer a hora?" com "Sim". Se quisermos saber que valor de *x* torna a sentença verdadeira, vamos precisar de uma função diferente, $A_{SK}V_{ARS}$, que chamamos com

$A_{SK}V_{ARS}$ (*BC*, Pessoa (x))

e que produz um fluxo de respostas. Nesse caso, haverá duas respostas: {*x*/ João} e {*x*/ Ricardo}. Tal resposta é chamada de **substituição** ou **lista de vinculação**. Normalmente, $A_{SK}V_{ARS}$ é reservado para bases de conhecimento constituídas exclusivamente por cláusulas de Horn, porque em tais bases todas as maneiras de tornar a consulta verdadeira irão vincular as variáveis a valores específicos. Não é o caso para a lógica de primeira ordem; se houver, na *BC*, *Rei*(*João*) $\lor$ *Rei*(*Ricardo*), não há nenhuma vinculação para *x* para a consulta $\exists$ *x Rei*(*x*), mesmo que a consulta seja verdadeira.

8.3.2 O domínio de parentesco

O primeiro exemplo que iremos considerar é o domínio de relacionamentos familiares ou de parentesco. Esse domínio inclui fatos como "Elizabeth é a mãe de Charles" e "Charles é o pai de William", e regras como "a avó de uma pessoa é a mãe do pai ou da mãe de uma pessoa".

É claro que os objetos em nosso domínio são pessoas. Teremos dois predicados unários, *Masculino* e *Feminino*. As relações de parentesco — paternidade, fraternidade, casamento, e assim por diante — serão representadas por predicados binários: *PaiOuMãe*, *IrmãoOuIrmã*, *Irmão*, *Irmã*, *FilhaOuFilho*, *Filha*, *Filho*, *Cônjuge*, *Esposa*, *Marido*, *AvôOuAvó*, *NetoOuNeta*, *Primo*, *Tia* e *Tio*.[9] Usaremos funções para representar *Mãe* e *Pai* porque toda pessoa tem exatamente um de cada um deles (pelo menos de acordo com o projeto da natureza).

Podemos acompanhar cada função e predicado anotando o que sabemos em termos dos outros símbolos. Por exemplo, a mãe de alguém é o pai ou mãe feminino deste alguém:

$\forall m, c\ Mãe(c) = m \Leftrightarrow Feminino(m) \wedge PaiOuMãe(m, c)$.

O marido de alguém é o cônjuge masculino de alguém:

$\forall w, h\ Marido(h, w) \Leftrightarrow Masculino(h) \wedge Cônjuge(h, w)$.

Masculino e feminino são categorias disjuntas:

$\forall x\ Masculino(x) \Leftrightarrow \neg Feminino(x)$.

PaiOuMãe e FilhoOuFilha são relações inversas:

$\forall p, c\ PaiOuMãe(p, c) \Leftrightarrow FilhoOuFilha(c, p)$.

Um Avô ou avó é pai ou mãe do pai ou da mãe de alguém:

$\forall g, c\ AvôOuAvó(g, c) \Leftrightarrow \exists p\ PaiOuMãe(g, p) \wedge PaiOuMãe(p, c)$.

Um irmão ou irmã é outro filho ou filha dos pais de alguém:

$\forall x, y\ IrmãoOuIrmã(x, y) \Leftrightarrow x \neq y \wedge \exists p\ PaiOuMãe(p, x) \wedge PaiOuMãe(p, y)$.

Poderíamos continuar por várias outras páginas como esse assunto; o Exercício 8.14 lhe pede para fazer exatamente isso.

Cada uma dessas sentenças pode ser vista como um **axioma** do domínio de parentesco, como explicado na Seção 7.1. Em geral, os axiomas estão associados a domínios puramente matemáticos — veremos em breve alguns axiomas referentes a números —, mas eles são necessários em todos os domínios. Os axiomas fornecem as informações factuais básicas a partir das quais podem ser derivadas conclusões úteis. Nossos axiomas de parentesco também são **definições**; eles têm a forma $\forall x,y\ P(x, y) \Leftrightarrow \ldots$. Os axiomas definem a função *Mãe* e os predicados *Marido*, *Masculino*, *PaiOuMãe*, *AvôOuAvó* e *IrmãoOuIrmã* em termos de outros predicados. Nossas definições se reduzem a um conjunto básico de predicados (*FilhoOuFilha*, *Cônjuge* e *Feminino*) em cujos termos os outros são definidos em última instância. Essa é uma forma muito natural de construir a representação de um domínio, e é análoga ao modo como os pacotes de software são elaborados por meio de definições sucessivas de sub-rotinas a partir de funções primitivas de biblioteca. Note que não existe necessariamente um conjunto único de predicados primitivos; poderíamos igualmente ter utilizado *PaiOuMãe*, *Cônjuge* e *Masculino*. Em alguns domínios, como mostraremos, não há nenhum conjunto básico claramente identificável.

Nem todas as sentenças lógicas sobre um domínio são axiomas. Algumas são **teoremas**, isto é, são consequência lógica dos axiomas. Por exemplo, considere a asserção de que a relação de irmãos é simétrico:

$\forall x, y\ IrmãoOuIrmã(x, y) \Leftrightarrow IrmãoOuIrmã(y, x)$.

Isso é um axioma ou um teorema? De fato, é um teorema que decorre logicamente do axioma que

define o parentesco. Se formularmos (com ASK) essa sentença à base de conhecimento, ela deverá retornar *verdadeiro*.

De um ponto de vista puramente lógico, uma base de conhecimento só precisa conter axiomas e não teoremas, porque os teoremas não aumentam o conjunto de conclusões que se seguem da base de conhecimento. Do ponto de vista prático, os teoremas são essenciais para reduzir o custo computacional da derivação de novas sentenças. Sem eles, um sistema de raciocínio tem de começar a partir de princípios fundamentais o tempo todo, como se fosse um físico obrigado a redefinir as regras do cálculo a cada novo problema.

Nem todos os axiomas são definições. Alguns fornecem informações mais gerais sobre certos predicados sem constituir uma definição. Na realidade, alguns predicados não têm nenhuma definição completa porque não sabemos o bastante para caracterizá-los plenamente. Por exemplo, não existe nenhuma forma óbvia definitiva de completar a sentença:

$\forall x\ Pessoa(x) \Leftrightarrow \ldots$

Felizmente, a lógica de primeira ordem nos permite fazer uso do predicado *Pessoa* sem defini-lo completamente. Em vez disso, podemos escrever especificações parciais de propriedades que toda pessoa tem e propriedades que tornam algo uma pessoa:

$\forall x\ Pessoa(x) \Rightarrow \ldots$
$\forall x\ \ldots \Leftarrow Pessoa(x).$

Os axiomas também podem ser "apenas fatos simples", como *Masculino*(*Jim*) e *Cônjuge*(*Jim, Laura*). Tais fatos formam as descrições de instâncias específicas de problemas, permitindo que perguntas específicas sejam respondidas. As respostas a essas perguntas serão então teoremas que decorrem dos axiomas. Com frequência, descobrimos que as respostas esperadas não estão disponíveis — por exemplo, de *Cônjuge*(*Jim, Laura*) espera-se (segundo as leis de muitos países) poder inferir ¬*Cônjuge*(*George, Laura*), mas isso não se origina dos axiomas dados anteriormente — mesmo depois de adicionarmos $Jim \neq George$, como sugerido na Seção 8.2.8. Esse é um sinal de que está faltando um axioma. O Exercício 8.8 lhe pede para fornecê-lo.

8.3.3 Números, conjuntos e listas

Os números talvez sejam o exemplo mais vívido de como uma grande teoria pode ser construída a partir de um minúsculo núcleo de axiomas. Descreveremos aqui a teoria dos **números naturais** ou inteiros não negativos. Precisaremos de um predicado *NumNat* que será verdadeiro sobre os números naturais, de um símbolo de constante, o 0, e ainda de um símbolo de função, *S* (sucessor). Os **axiomas de Peano** definem números naturais e a adição.[10] Os números naturais são definidos recursivamente:

$NumNat(0).$
$\forall n\ NumNat(n) \Rightarrow NumNat(S(n)).$

Isto é, 0 é um número natural e, para todo objeto n, se n é um número natural, então $S(n)$ é um número natural. Assim, os números naturais são 0, $S(0)$, $S(S(0))$, e assim por diante. (Depois de ler a Seção 8.2.8, você vai notar que esses axiomas permitem outros números naturais, além dos usuais; veja o Exercício 8.12.) Também precisamos de axiomas para restringir a função sucessora:

$\forall n\ 0 \neq S(n)$.
$\forall m, n\ m \neq n \Rightarrow S(m) \neq S(n)$.

Agora, podemos definir a adição em termos da função sucessora:

$\forall m\ NumNat(m) \Rightarrow +(0, m) = m$.
$\forall m, n\ NumNat(m) \land NumNat(n) \Rightarrow +(S(m), n) = S(+(m, n))$.

O primeiro desses axiomas afirma que a adição de 0 a qualquer número natural m fornece o próprio m. Note o uso do símbolo de função binária "+" no termo $+(m, 0)$; em matemática comum, o termo seria escrito como $m + 0$ usando-se a notação **infixa** (a notação que usamos na lógica de primeira ordem é chamada **prefixa**).

Para facilitar a leitura de nossas sentenças sobre números, permitiremos o uso da notação infixa. Também escreveremos $S(n)$ como $n + 1$, de forma que o segundo axioma se torne:

$\forall m, n\ NumNat(m) \land NumNat(n) \Rightarrow (m + 1) + n = (m + n) + 1$.

Esse axioma reduz a adição à aplicação repetida da função sucessora.

O uso da notação infixa é um exemplo de **açúcar sintático**, ou seja, uma extensão para a nossa abreviação da sintaxe-padrão que não altera a semântica. Qualquer sentença que utilize açúcar pode ser "desaçucarada" para produzir uma sentença equivalente em lógica de primeira ordem comum.

Uma vez que temos a adição, é simples definir a multiplicação como a adição repetida, a exponenciação como a multiplicação repetida, a divisão de inteiros e restos, números primos, e assim por diante. Desse modo, toda a teoria dos números (inclusive a criptografia) pode ser elaborada a partir de uma constante, uma função, um predicado e quatro axiomas.

O domínio de **conjuntos** também é fundamental para a matemática, bem como para o raciocínio comum (de fato, é possível definir a teoria dos números em termos da teoria dos conjuntos). Queremos ter a capacidade de representar conjuntos individuais, inclusive o conjunto vazio. Precisamos de um modo de construir conjuntos acrescentando um elemento a um conjunto ou tomando a união ou interseção de dois conjuntos. Queremos saber se um elemento pertence a um conjunto e queremos ter a possibilidade de distinguir conjuntos de objetos que não são conjuntos.

Utilizaremos o vocabulário normal da teoria dos conjuntos como açúcar sintático. O conjunto vazio é uma constante escrita como { }. Existe um único predicado unário, *Conjunto*, que é verdadeiro no caso de conjuntos. Os predicados binários são $x \in s$ (x pertence ao conjunto s) e $s_1 \subseteq s_2$ (o conjunto s_1 é um subconjunto, não necessariamente próprio, do conjunto s_2). As funções binárias são $s_1 \cap s_2$ (a interseção de dois conjuntos), $s_1 \cup s_2$ (a união de dois conjuntos) e $\{x|s\}$ (o conjunto resultante da adição do elemento x ao conjunto s). Um conjunto possível de axiomas é:

1. Apenas são conjuntos o conjunto vazio e os que são formados pela adição de algo a um conjunto:

$$\forall s \; Conjunto(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_2 \; Conjunto(s_2) \wedge s = (x|s_2\}).$$

2. O conjunto vazio não tem elementos adicionados a ele; em outras palavras, não há como decompor { } em um conjunto menor e um elemento:

$$\neg\exists x, s \; \{x|s\} = \{ \}.$$

3. A adição de um elemento que já está no conjunto não tem nenhum efeito:

$$\forall x, s \; x \in s \Leftrightarrow s = \{x|s\}.$$

4. Os únicos membros de um conjunto são os elementos que foram adicionados a ele. Expressamos esse fato de modo recursivo, afirmando que x é um membro de s (ou pertence a s) se e somente se s é igual a algum conjunto s_2 adicionado a algum elemento y, onde y é igual a x ou x é um membro de s_2:

$$\forall x, s \; x \in s \Leftrightarrow \exists y, s_2 \; (s = \{y|s_2\} \wedge (x = y \vee x \in s_2)).$$

5. Um conjunto é um subconjunto de outro conjunto se e somente se todos os elementos do primeiro conjunto pertencem ao segundo conjunto:

$$\forall s_1, s_2 \; s_1 \subseteq s_2 \Leftrightarrow (\forall s \; x \in s_1 \Rightarrow x \in s_2).$$

6. Dois conjuntos são iguais se e somente se cada um deles é um subconjunto do outro:

$$\forall s_1, s_2 \; (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s2 \wedge s_2 \subseteq s_1).$$

7. Um objeto está na interseção de dois conjuntos se e somente se ele pertence a ambos os conjuntos:

$$\forall x, s_1, s_2 \; x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2).$$

8. Um objeto está na união de dois conjuntos se e somente se ele pertence a um ou a ambos os conjuntos:

$$\forall x, s_1, s_2 \; x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2).$$

As **listas** são semelhantes aos conjuntos. As diferenças residem no fato de que as listas são ordenadas e de que o mesmo elemento pode aparecer mais de uma vez em uma lista. Podemos usar o vocabulário de Lisp para listas:

Nil é a lista constante, sem elementos; *Cons*, *Append*, *First* e *Rest* são funções e *Find* é o predicado que representa para as listas aquilo que *Membro* representa para conjuntos. *List*? é um predicado verdadeiro apenas no caso de listas. Como ocorre com os co*n*juntos, é comum usar açúcar

sintático em sentenças lógicas que envolvem listas. A lista vazia é representada por [ ]. O termo *Cons*(x, y), onde y é uma lista não vazia, é escrito como [x|y]. O termo *Cons*(x, *Nil*) — isto é, a lista que contém o elemento x — é escrito como [x]. Uma lista de vários elementos, como [A, B, C], corresponde ao termo aninhado *Cons*(A, *Cons*(B, *Cons*(C, *Nil*))). O Exercício 8.16 lhe pede para relacionar os axiomas referentes a listas.

8.3.4 O mundo de wumpus

Alguns axiomas de lógica proposicional para o mundo de wumpus foram dados no Capítulo 7. Os axiomas de primeira ordem desta seção são muito mais concisos, captando de modo muito natural exatamente aquilo que queremos dizer.

Lembre-se de que o agente de wumpus recebe um vetor de percepções com cinco elementos. A sentença de primeira ordem correspondente armazenada na base de conhecimento deve incluir tanto a percepção quanto a instante em que ela ocorreu; caso contrário, o agente ficará confuso sobre o momento em que viu cada item. Utilizaremos os inteiros como instantes temporais. Uma sentença de percepções típica seria:

Percepção([*Fedor*, *Brisa*, *Brilho*, *Nenhum*, *Nenhum*], 5).

Aqui, *Percepção* é um predicado binário; *Fedor,* e assim por diante, são constantes inseridas em uma lista. As ações no mundo de wumpus podem ser representadas por termos lógicos:

Virar(*Direita*), *Virar*(*Esquerda*), *Avançar*, *Atirar*, *Agarrar*, *Soltar*, *Escalar.*

Para determinar o que é melhor, o programa do agente constrói uma consulta como:

ASKVARS (∃ *a MelhorAção*(*a*, 5)),

que retorna uma lista de vinculação como {*a*/*Agarrar*}. O programa do agente pode então retornar *Agarrar* como a ação a executar. Os dados brutos da percepção implicam certos fatos sobre o estado atual. Por exemplo,

∀ *t, s, g, m, c Percepção*([*s*, *Brisa, g, m, c*], *t*) ⇒ *Brisa* (*t*),
∀ *t, s, b, m, c Percepção*([*s*, *b*, *Brilho, m, c*], *t*) ⇒ *Brilho*(*t*),

e assim por diante. Essas regras exibem uma forma trivial do processo de raciocínio chamado **percepção**, que estudaremos em profundidade no Capítulo 24. Note a quantificação sobre o tempo *t*. Em lógica proposicional, precisaríamos de cópias de cada sentença para cada instante de tempo.

O comportamento "reativo" simples também pode ser implementado por sentenças de implicação quantificadas. Por exemplo, temos:

∀ *t Brilho*(*t*) ⇒ *MelhorAção*(*Agarrar, t*).

Dadas a percepção e as regras dos parágrafos precedentes, isso produziria a conclusão desejada *MelhorAção*(*Agarrar*, 5), ou seja, *Agarrar* é a ação correta.

Representamos as entradas e saídas do agente; agora, chegou a hora de representar o próprio ambiente. Vamos começar com os objetos. Candidatos óbvios são quadrados, poços e o wumpus. Poderíamos nomear cada quadrado — $Quadrado_{1,2}$ e assim por diante —, entretanto, o fato de $\text{Quadrado}_{1,2}$ e $Quadrado_{1,3}$ serem adjacentes teria de ser um fato "extra", e precisaríamos de um fato desse tipo para cada par de quadrados. É melhor usar um termo complexo em que a linha e a coluna aparecem como inteiros; por exemplo, podemos simplesmente utilizar o termo lista [1, 2]. A adjacência de dois quadrados quaisquer pode ser definida como:

$$\forall x, y, a, b\ Adjacente([x, y], [a, b]) \Rightarrow$$
$$(x = a \land (y = b - 1 \lor y = b + 1)) \lor (y = b \land (x = a - 1 \lor x = a + 1)).$$

Também poderíamos nomear cada poço, mas isso seria inadequado por uma razão diferente: não há nenhum motivo para fazer distinção entre os poços.[11] É muito mais simples usar um predicado unário *Poço* que seja verdadeiro no caso de quadrados contendo poços. Por fim, tendo em vista que há exatamente um wumpus, uma constante *Wumpus* é um predicado unário muito bom (e talvez mais digno, do ponto de vista do wumpus).

A posição do agente muda com o tempo, e assim escreveremos *Em*(*Agente*, *s*, *t*) para indicar que o agente está no quadrado *s* no instante *t*. Podemos fixar a localização do wumpus com $\forall t$ *Em*(*Wumpus*, [2, 2], *t*).

Podemos então dizer que os objetos só podem estar em um local de cada vez:

$$\forall x, s_1, s_2, t\ At(x, s_1, t) \land At(x, s_2, t) \Rightarrow s_1 = s_2.$$

Dada sua posição atual, o agente pode deduzir propriedades do quadrado a partir de propriedades de sua percepção atual. Por exemplo, se o agente estiver em um quadrado e perceber uma brisa, então esse quadrado é arejado:

$$\forall s, t\ Em(Agente, s, t) \land Brisa(t) \Rightarrow Arejado(s).$$

É útil saber que um *quadrado* é arejado porque sabemos que os poços não podem se deslocar. Note que *Arejado* não tem nenhum argumento de tempo.

Tendo descoberto quais são os lugares arejados (ou fedorentos) e, muito importante, os lugares *não* arejados (ou *não* fedorentos), o agente pode deduzir onde estão os poços (e onde está o wumpus). Enquanto a lógica proposicional necessita de um axioma separado para cada quadrado (veja R_2 e R_3 na Seção 7.4.3) e precisaria de um conjunto de axiomas diferentes para cada configuração geográfica do mundo, a lógica de primeira ordem precisa apenas de um axioma:

$$\forall s Arejado(s) \Leftrightarrow \exists r Adjacente(r, s) \land Poço(r). \quad (8.4)$$

Da mesma forma, na lógica de primeira ordem podemos quantificar ao longo do tempo, e assim precisamos apenas de um axioma de estado sucessor para cada predicado, em vez de uma cópia

diferente para cada instante. Por exemplo, o axioma para Flecha (Equação 7.2) torna-se

$$\forall t\ TemFlecha\,(t+1) \Leftrightarrow (TemFlecha(t)\ \wedge \neg Ação\,(Atirar,\, t)).$$

A partir dessas duas sentenças de exemplo, podemos ver que a formulação da lógica de primeira ordem não é menos concisa do que a descrição original em linguagem natural dada no Capítulo 7. O leitor é convidado a construir axiomas análogos para a localização e orientação do agente; nesses casos, os axiomas quantificam sobre espaço e tempo. Como no caso da estimação do estado proposicional, um agente pode usar inferência lógica com os axiomas desse tipo para manter o controle de aspectos do mundo que não são observados diretamente. No Capítulo 10 nos aprofundaremos mais sobre o tema dos axiomas de estado sucessor de primeira ordem e seus usos para a construção de planos.

8.4 ENGENHARIA DE CONHECIMENTO EM LÓGICA DE PRIMEIRA ORDEM

A seção precedente ilustrou o uso da lógica de primeira ordem para representar o conhecimento em três domínios simples. Esta seção descreve o processo geral de construção da base de conhecimento — um processo chamado **engenharia de conhecimento**. Um engenheiro de conhecimento é alguém que investiga um domínio específico, aprende quais conceitos são importantes nesse domínio e cria uma representação formal dos objetos e relações no domínio. Ilustraremos o processo de engenharia de conhecimento em um domínio de circuito eletrônico que já deve ser bastante familiar; dessa forma, poderemos nos concentrar nas questões de representação envolvidas. A abordagem que adotaremos é adequada para o desenvolvimento de bases de conhecimento de *uso especial*, cujo domínio está cuidadosamente circunscrito e cujo intervalo de consultas é conhecido com antecedência. As bases de conhecimento de *uso geral*, destinadas a dar suporte a consultas em toda a variedade do conhecimento humano e a apoiar tarefas tais como a compreensão da linguagem natural, serão descritas no Capítulo 12.

8.4.1 O processo de engenharia de conhecimento

Os projetos de engenharia de conhecimento variam amplamente em conteúdo, escopo e dificuldade, mas todos eles incluem as etapas a seguir:

1. *Identificar a tarefa*. O engenheiro de conhecimento deve delinear a variedade de questões que a base de conhecimento admitirá e os tipos de fatos que estarão disponíveis para cada instância específica de problema.

   Por exemplo, a base de conhecimento do wumpus precisa ser capaz de escolher ações ou ela é obrigada a responder a questões apenas sobre o conteúdo do ambiente? Os fatos do sensor incluirão a posição atual? A tarefa determinará que conhecimento deve ser representado, com a finalidade de conectar instâncias de problemas a respostas. Essa etapa é análoga ao processo

PEAS para projetar agentes, que vimos no Capítulo 2.

2. *Agregar o conhecimento relevante*. O engenheiro de conhecimento já deve ser um especialista no domínio ou talvez precise trabalhar com especialistas reais para extrair o que eles conhecem — um processo chamado **aquisição de conhecimento**. Nessa fase, o conhecimento não é representado formalmente. A ideia é compreender o escopo da base de conhecimento, determinada pela tarefa, e entender como o domínio realmente funciona.

   No caso do mundo de wumpus, definido por um conjunto artificial de regras, é fácil identificar o conhecimento relevante (contudo, note que a definição de adjacência não foi fornecida explicitamente nas regras do mundo de wumpus). Para domínios reais, a questão de relevância pode ser bastante difícil — por exemplo, um sistema para simular projetos VLSI pode necessitar levar em conta ou não capacitâncias parasitas e efeitos peliculares.

3. *Definir um vocabulário de predicados*, *funções* e *constantes*. Ou seja, converter os importantes conceitos de nível de domínio em nomes no nível de lógica. Isso envolve muitas questões de *estilo* da engenharia de conhecimento. Como o estilo de programação, ele pode ter um impacto significativo sobre o sucesso final do projeto. Por exemplo, os poços devem ser representados por objetos ou por um predicado unário sobre quadrados? A orientação do agente deve ser uma função ou um predicado? A posição do wumpus deve depender do tempo? Uma vez que as escolhas tenham sido feitas, o resultado é um vocabulário conhecido como **ontologia** do domínio. A palavra *ontologia* representa uma teoria específica sobre a natureza de ser ou existir. A ontologia determina os tipos de itens que existem, mas não determina suas propriedades específicas e seus inter-relacionamentos.

4. *Codificar o conhecimento geral sobre o domínio*. O engenheiro de conhecimento escreve os axiomas correspondentes a todos os termos do vocabulário. Isso fixa (na medida do possível) o significado dos termos, permitindo ao especialista verificar o conteúdo. Com frequência, essa etapa revela concepções erradas ou lacunas no vocabulário que devem ser corrigidas retornando à etapa 3 e repetindo-se todo o processo.

5. *Codificar uma descrição da instância específica do problema*. Se a ontologia estiver bem elaborada, essa etapa será fácil. Ela envolverá a escrita de sentenças atômicas simples sobre instâncias de conceitos que já fazem parte da ontologia. Para um agente lógico, as instâncias de problemas são fornecidas pelos sensores, enquanto uma base de conhecimento "desincorporada" é suprida com sentenças adicionais, da mesma forma que os programas tradicionais são supridos com dados de entrada.

6. *Formular consultas ao procedimento de inferência e obter respostas*. Essa é a etapa em que está a recompensa: podemos deixar o procedimento de inferência operar sobre os axiomas e fatos específicos do problema para derivar os fatos que estamos interessados em conhecer. Assim, evitamos a necessidade de escrever um algoritmo para solução de aplicação específica.

7. *Depurar a base de conhecimento*. Infelizmente, as respostas às consultas poucas vezes estarão corretas na primeira tentativa. Mais precisamente, as respostas estarão corretas *para a base de conhecimento criada*, supondo-se que o procedimento de inferência seja consistente, mas não

serão aquelas que o usuário espera. Por exemplo, se um axioma estiver ausente, algumas consultas não poderão ser respondidas a partir da base de conhecimento. Disso poderia resultar um processo de depuração considerável. Falta de axiomas ou axiomas muito fracos pode ser identificado com facilidade observando-se lugares em que a cadeia de raciocínio se interrompe de forma inesperada. Por exemplo, se a base de conhecimento incluir uma regra de diagnóstico (veja o Exercício 8.13) para encontrar o wumpus,

$\forall s\ Fedor(r) \Rightarrow Adjacente(Casa(Wumpus), \mathrm{s})$,

em vez de bicondicional, o agente nunca poderá provar a *ausência* de wumpuses. Os axiomas incorretos podem ser identificados porque são declarações falsas sobre o mundo. Por exemplo, a sentença

$\forall x\ NumDePernas(x, 4) \Rightarrow Mamífero(x)$

 é falsa para répteis, anfíbios e, mais importante, mesas. *A falsidade dessa sentença pode ser determinada de forma independente do restante da base de conhecimento*. Em contraste, um erro típico em um programa é semelhante a:

deslocamento = posição + 1.

É impossível dizer se essa declaração é correta sem examinar o restante do programa para ver se, por exemplo, o deslocamento é usado para se referir à posição atual ou a uma unidade além da posição atual ou se o valor de posição é alterado por outra declaração e, portanto, se o deslocamento também deve ser mais uma vez alterado.

Para entender melhor esse processo de sete etapas, vamos aplicá-lo agora a um exemplo ampliado — o domínio de circuitos eletrônicos.

8.4.2 O domínio de circuitos eletrônicos

Desenvolveremos uma ontologia e uma base de conhecimento que nos permitirão raciocinar sobre circuitos digitais do tipo mostrado na Figura 8.6. Seguiremos o processo de sete etapas da engenharia de conhecimento.

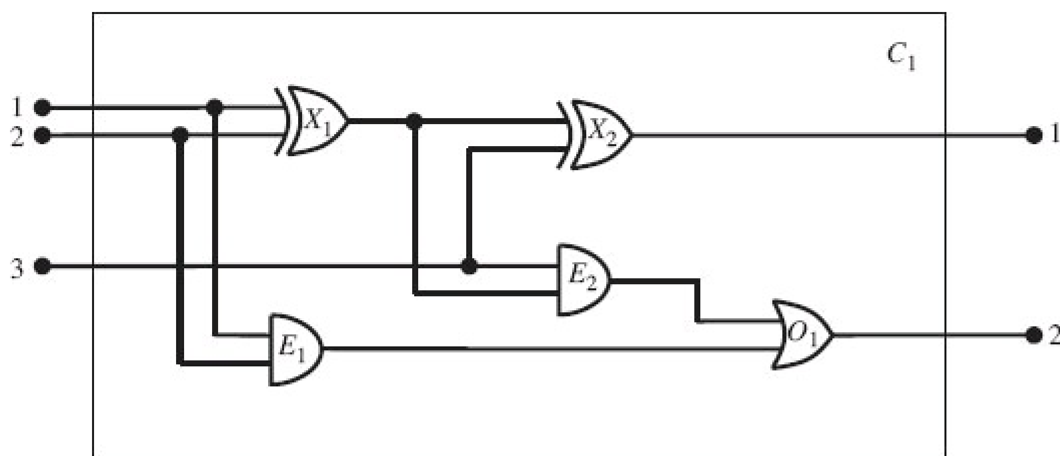


Figura 8.6 Um circuito digital C1, que deve ser um somador completo de um bit. As duas primeiras entradas são os dois bits a serem somados, e a terceira entrada é um bit de transporte. A primeira saída é a soma, e a segunda saída é um bit de transporte para o próximo somador. O circuito contém duas portas XOU (ou exclusivo), duas portas E e uma porta OU. mais fácil de ler.

Identificar a tarefa

Existem muitas tarefas de raciocínio associadas a circuitos digitais. No nível mais alto, analisamos a funcionalidade do circuito. Por exemplo, o circuito na Figura 8.6 realmente efetua soma de modo apropriado? Se todas as entradas são altas, qual será a saída da porta A2? Perguntas sobre a estrutura do circuito também são interessantes. Por exemplo, quais são as portas conectadas ao primeiro terminal de entrada? O circuito contém laços de realimentação? Essas serão nossas tarefas nesta seção. Existem níveis mais detalhados de análise, incluindo aqueles relacionados a retardos de sincronização, área de circuitos, consumo de energia, custo de produção, e assim por diante. Cada um desses níveis exigiria conhecimento adicional.

Agregar o conhecimento relevante

O que sabemos sobre circuitos digitais? Para nossos propósitos, eles são formados por fios e portas. Os sinais fluem pelos fios até os terminais de entrada das portas, e cada porta produz um sinal no terminal de saída que flui por outro fio. Para determinar quais serão esses sinais, precisamos saber como as portas transformam seus sinais de entrada. Existem quatro tipos de portas: as portas E, OU e XOU têm dois terminais de entrada, e as portas NÃO têm um. Todas as portas têm um terminal de saída. Os circuitos, como as portas, têm terminais de entrada e de saída.

Para raciocinar sobre funcionalidade e conectividade, não precisamos mencionar os próprios fios, os caminhos que eles seguem ou as junções em que eles se unem. Tudo o que importa são as conexões entre terminais — podemos dizer que um terminal de saída está conectado a outro terminal de entrada sem ter de mencionar o fio que realmente os conecta. Outros fatores, tais como tamanho, forma, cor ou o custo dos vários componentes, são irrelevantes para nossa análise.

Se nosso propósito fosse algo diferente de verificar projetos no nível de porta, a ontologia seria diferente. Por exemplo, se estivéssemos interessados em depurar circuitos defeituosos, provavelmente seria boa ideia incluir os fios na ontologia porque um fio defeituoso pode corromper o sinal que flui através dele. Para solucionar falhas de sincronismo, precisaríamos incluir retardos de portas. Se estivéssemos interessados em projetar um produto que fosse lucrativo, o custo do circuito e sua velocidade relativa a outros produtos no mercado seriam fatores importantes.

Definir um vocabulário

Agora sabemos que desejamos abordar circuitos, terminais, sinais e portas. A próxima etapa é escolher funções, predicados e constantes para representá-los. Primeiro, precisamos ser capazes de distinguir as portas umas das outras e de outros objetos. Cada porta é representada como um objeto denominado por uma constante, sobre a qual afirmamos que é uma porta como, digamos, $Porta(X_1)$. O comportamento de cada porta é determinado pelo seu tipo: uma das constantes E, OU, XOU ou NÃO. Devido à porta ter exatamente um tipo, é apropriada uma função: $Tipo(X_1) = XOU$. Circuitos, como portas, são identificados por um predicado: $Circuito(C_1)$.

Em seguida, consideraremos os terminais, que estão identificados pelo predicado $Terminal(x)$.Uma porta ou um circuito pode ter um ou mais terminais de entrada e um ou mais terminais de saída. Utilizamos a função $Entrada(1, X_1)$ para denotar o primeiro terminal de entrada para a porta X_1. Uma função similar *Saída* é utilizada para terminais de saída. A função $Aridade(c, i, j)$ diz que o circuito c tem i terminais de entrada e j terminais de saída. A conectividade entre portas pode ser representada pelo predicado *Conectado*, que recebe dois terminais como argumentos, como em $Conectado(Saída(1, X_1), Entrada(1, X_2))$.

Finalmente, precisamos saber se um sinal está ligado ou desligado. Uma possibilidade é usar um predicado unário, $On(t)$, que é verdadeiro quando o sinal em um terminal está ligado. No entanto, isso torna um pouco difícil a formulação de perguntas como: "Quais são todos os valores possíveis dos sinais nos terminais de saída do circuito C_1?" Portanto, introduziremos como objetos dois "valores de sinal", 1 e 0, e uma função $Sinal(t)$ que denota o valor de sinal para o terminal t.

Codificar o conhecimento geral do domínio

Um sinal de que temos uma boa ontologia é que necessitamos apenas de umas poucas regras gerais, que podem ser expressas clara e concisamente. Esses são os axiomas necessários:

1. Se dois terminais estão conectados, eles têm o mesmo sinal:

$$\forall t_1, t_2\ Terminal\ (t_1) \land Terminal\ (t_2) \land Conectado(t_1, t_2) \Rightarrow Sinal\ (t_1) = Sinal\ (t_2).$$

2. O sinal em todo terminal é ou 1 ou 0:

$$\forall t\ Terminal\ (t) \Rightarrow Sinal\ (t)=1 \lor Sinal\ (t)=0.$$

3. Conectado é um predicado comutativo:

$$\forall t_1, t_2\ Conectado(t_1, t_2) \Leftrightarrow Conectado(t_2, t_1).$$

4. Há quatro tipos de portas:

$$\forall \mathrm{g}\ Porta(g) \land k = Tipo(g) \Rightarrow k = E \lor k = OU \lor k = XOU \lor k = NÃO.$$

5. A saída de uma porta E é 0 se e somente se qualquer de suas entradas for 0:

$\forall$ g $Porta$(g) $\wedge$ $Tipo$(g)= E $\Rightarrow$
$Sinal(Saída(1, g)) = 0 \Leftrightarrow \exists n\ Sinal(Entrada(n, g)) = 0$.

6. A saída de uma porta OU é 1 se e somente se quaisquer de suas saídas for 1:

$\forall$ g $Porta(g) \wedge Tipo(g) = OU \Rightarrow$
$Sinal(Saída(1, g)) = 1 \Leftrightarrow \exists n\ Sinal(Entrada(n, g)) = 1$.

7. A saída de uma porta XOU é 1 se e somente se suas entradas forem diferentes:

$\forall$ g $Porta(g) \wedge Tipo(g) = XOU \Rightarrow$
$Sinal(Saída(1, g)) = 1 \Leftrightarrow Sinal(Entrada(1, g) \neq Sinal(Entrada(2, g))$.

8. A saída de uma porta NÃO é diferente de sua entrada:

$\forall$ g $Porta(g) \wedge Tipo(g) = NÃO \Rightarrow$
$Sinal(Saída(1, g)) \neq Sinal(Entrada(1, g))$.

9. As portas (exceto para NÃO) têm duas entradas e uma saída.

$\forall$ g $Porta(g) \wedge Tipo(g) = NÃO \Rightarrow Aridade\ (g,1,1)$.
$\forall$ g $Porta(g) \wedge k = Tipo(g) \wedge (k = E \vee k = OU \vee k = XOU) \Rightarrow$
$Aridade\ (g,2,1)$.

10. Um circuito tem terminais, até sua aridade de entrada e saída, e nada além de sua aridade:

$\forall c, i, j\ Circuito(c) \wedge Aridade(c, i, j) \Rightarrow$
$\forall n\ (n \leq i \Rightarrow Terminal\ (Entrada(c, n))) \wedge (n > i \Rightarrow Entrada(c, n) = Nada) \wedge$
$\forall n\ (n \leq j \Rightarrow Terminal\ (Saída(c, n))) \wedge (n > j \Rightarrow Saída(c, n) = Nada)$.

11. Todos são distintos: portas, terminais, signais, tipos de porta e *Nada*.

$\forall g, t\ Porta(g) \wedge Terminal\ (t) \Rightarrow$
$g \neq t \neq 1 \neq 0 \neq OU \neq E \neq XOU \neq NÃO \neq Nada$.

12. Portas são circuitos.

$\forall$ g $Porta(g) \Rightarrow Circuito(g)$.

Codificar a instância específica do problema

O circuito mostrado na Figura 8.6 é codificado como o circuito C_1 com a descrição a seguir. Primeiro, categorizamos o circuito e suas portas componentes:

$$Circuito(C_1) \wedge Aridade(C_1, 3, 2)$$
$$Porta(X_1) \wedge Tipo(X_1)=XOU$$
$$Porta(X_2) \wedge Tipo(X_2)=XOU$$
$$Porta(A_1) \wedge Tipo(A_1)=E$$
$$Porta(A_2) \wedge Tipo(A_2)=E$$
$$Porta(O_1) \wedge Tipo(O_1)=OU.$$

Em seguida, mostramos as conexões entre elas:

$$Conectado(Saída(1, X_1), Entrada(1, X_2)) \quad Conectado(Entrada(1, C_1), Entrada(1, X_1))$$
$$Conectado(Saída(1, X_1), Entrada(2, A_2)) \quad Conectado(Entrada(1, C_1), Entrada(1, A_1))$$
$$Conectado(Saída(1, A_2), Entrada(1, O_1)) \quad Conectado(Entrada(2, C_1), Entrada(2, X_1))$$
$$Conectado(Saída(1, A_1), Entrada(2, O_1)) \quad Conectado(Entrada(2, C_1), Entrada(2, A_1))$$
$$Conectado(Saída(1, X_2), Saída(1, C_1)) \quad Conectado(Entrada(3, C_1), Entrada(2, X_2))$$
$$Conectado(Saída(1, O_1), Saída(2, C_1)) \quad Conectado(Entrada(3, C_1), Entrada(1, A_2)).$$

Formular consultas ao procedimento de inferência

Que combinações de entradas fariam a primeira saída de C_1 (o bit de soma) ser 0 e a segunda saída de C_1 (o bit de transporte) ser 1?

$$\exists i_1, i_2, i_3 \; Sinal(Entrada(1, C_1)) = i_1 \wedge Sinal(Entrada(2, C_1)) = i_2$$
$$\wedge \; Sinal(Entrada(3, C_1)) = i_3 \wedge Sinal(Saída(1, C_1)) = 0 \wedge Sinal(Saída(2, C_1)) = 1 \; .$$

As respostas são substituições para as variáveis $i1$, $i2$ e $i3$ tais que a sentença resultante é consequência lógica da base de conhecimento. ASKVARS vai fornecer três substituições desse tipo:

$$\{i_1/1, i_2/1, i_3/0\} \; \{ i_1/1, i_2/0, i_3/1\} \; \{ i_1/0, i_2/1, i_3/1\} \; .$$

Quais são os conjuntos de valores possíveis de todos os terminais para o circuito somador?

$$\exists i_1, i_2, i_3, o_1, o_2 \; Sinal(Entrada(1, C_1)) = i_1 \wedge Sinal(Entrada(2, C_1)) = i_2$$
$$\wedge \; Sinal(Entrada(3, C_1)) = i_3 \wedge Sinal(Saída(1, C_1)) = o_1 \wedge Sinal(Saída(2, C_1)) = o_2 \; .$$

Essa consulta final retornará uma tabela completa de entrada/saída para o dispositivo, que poderá ser usada para verificar se de fato o circuito soma suas entradas de maneira correta. Esse é um exemplo simples de **verificação de circuitos**. Também podemos usar a definição de circuito para elaborar sistemas digitais maiores, para os quais é possível executar o mesmo tipo de procedimento de verificação (veja o Exercício 8.26). Muitos domínios são apropriados para o mesmo tipo de desenvolvimento estruturado de bases de conhecimento, no qual conceitos mais complexos são definidos a partir de conceitos mais simples.

Depurar a base de conhecimento

Podemos perturbar a base de conhecimento de várias maneiras, a fim de verificar que tipos de comportamentos errôneos emergem. Por exemplo, suponha que deixemos de ler a Seção 8.2.8 e, portanto, nos esqueçamos da asserção de que $1 \neq 0$. De repente, o sistema será incapaz de provar

quaisquer saídas para o circuito, com exceção dos casos de entrada 000 e 110. Podemos identificar o problema solicitando as saídas de cada porta. Por exemplo, podemos solicitar:

$$\exists\, i_1, i_2, o\ Sinal(Entrada(1, C_1)) = i_1 \land Sinal(Entrada(2, C_1)) = i_2 \land Sinal(Saída(1, X_1)) = o,$$

o que revela que nenhuma saída é conhecida em X_1 nos casos de entrada 10 e 01. Em seguida, observamos o axioma para portas XOU, aplicado a X_1:

$$Sinal(Saída(1, X_1) = 1 \Leftrightarrow Sinal(Entrada(1, X_1)) \neq Sinal(Entrada(2, X_1)).$$

Se sabemos que as entradas são, digamos, 1 e 0, isso se reduz a:

$$Sinal(Saída(1, X_1)) = 1 \Leftrightarrow 1 \neq 0.$$

Agora o problema é aparente: o sistema é incapaz de deduzir que $Sinal(Saída(1, X_1)) = 1$, e assim precisamos informá-lo de que $1 \neq 0$.

8.5 RESUMO

Este capítulo introduziu a **lógica de primeira ordem**, uma linguagem de representação muito mais poderosa que a lógica proposicional. Os pontos importantes são:

- As linguagens de representação do conhecimento devem ser declarativas, composicionais, expressivas, independentes do contexto e não ambíguas.
- As lógicas diferem em seus **compromissos ontológicos** e **compromissos epistemológicos**. Embora a lógica proposicional só se comprometa com a existência de fatos, a lógica de primeira ordem se compromete com a existência de objetos e relações, e, portanto, ganha expressividade.
- A sintaxe da lógica de primeira ordem baseia-se na da lógica proposicional. Acrescenta termos para representar objetos e tem quantificadores universais e existenciais para construir asserções sobre todos ou alguns dos valores possíveis das variáveis quantificadas.
- Um **mundo possível**, ou **modelo**, para a lógica de primeira ordem inclui um conjunto de objetos e uma **interpretação** que projeta símbolos constantes para objetos, símbolos de predicados para as relações entre os objetos e símbolos de função para as funções sobre os objetos.
- Uma sentença atômica será verdadeira apenas quando a relação denominada pelo predicado se mantiver entre os objetos denominados pelos termos. As **interpretações estendidas**, que projetam variáveis quantificadoras a objetos no modelo, definem a verdade das sentenças quantificadas.
- O desenvolvimento de uma base de conhecimento em lógica de primeira ordem exige um processo cuidadoso de análise do domínio, escolha de um vocabulário e codificação dos axiomas necessários para dar suporte às inferências desejadas.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Embora até mesmo a lógica de Aristóteles lide com generalizações a respeito de objetos, ficou muito aquém do poder expressivo da lógica de primeira ordem. Uma grande barreira ao seu desenvolvimento posterior foi a sua concentração em predicados unários, excluindo predicados relacionais de maior aridade. O primeiro tratamento sistemático das relações foi dada por Augustus De Morgan (1864), que citou o exemplo seguinte para mostrar os tipos de inferências com as quais a lógica de Aristóteles não poderia lidar: "Todos os cavalos são animais; portanto, a cabeça de um cavalo é a cabeça de um animal." Essa inferência é inacessível para Aristóteles porque qualquer regra válida que possa apoiá-la deve primeiro analisar a sentença utilizando o predicado binário "x é a cabeça de y". A lógica das relações foi estudada em profundidade por Charles Sanders Peirce (1870, 2004).

A verdadeira lógica de primeira ordem data da introdução de quantificadores no trabalho de Gottlob Frege (1879), *Begriffschrift* ("Escrita de Conceitos" ou "Notação Conceitual"). Peirce (1883) também desenvolveu a lógica de primeira ordem independentemente de Frege, apesar de um pouco mais tarde. A habilidade de Frege para aninhar quantificadores foi um grande passo à frente, mas ele utilizava uma notação desajeitada. A notação atual para a lógica de primeira ordem se deve substancialmente a Giuseppe Peano (1889), mas a semântica é virtualmente idêntica à de Frege. Por estranho que pareça, os axiomas de Peano devem ser creditados em grande parte a Grassmann (1861) e Dedekind (1888).

Leopold Löwenheim (1915) deu um tratamento sistemático à teoria dos modelos para a lógica de primeira ordem, incluindo o primeiro tratamento apropriado do símbolo de igualdade. Os resultados de Löwenheim foram estendidos mais ainda por Thoralf Skolem (1920). Alfred Tarski (1935, 1956) deu uma definição explícita de verdade e de satisfação da teoria de modelos em lógica de primeira ordem, utilizando a teoria de conjuntos.

McCarthy (1958) foi o principal responsável pela introdução da lógica de primeira ordem como uma ferramenta para a construção de sistemas de IA. As perspectivas da IA baseada em lógica tiveram um avanço significativo a partir do desenvolvimento por Robinson (1965) da resolução, um procedimento completo para inferência de primeira ordem, descrito no Capítulo 9. A abordagem logicista criou raízes na Universidade de Stanford. Cordell Green (1969a, 1969b) desenvolveu um sistema de raciocínio de primeira ordem, o QA3, levando às primeiras tentativas de construir um robô lógico no SRI (Fikes e Nilsson, 1971). A lógica de primeira ordem foi aplicada por Zohar Manna e Richard Waldinger (1971) ao raciocínio sobre programas e, mais tarde, por Michael Genesereth (1984) ao raciocínio sobre circuitos. Na Europa, a programação em lógica (uma forma restrita de raciocínio de primeira ordem) foi desenvolvida para análise linguística (Colmerauer *et al*., 1973) e para sistemas declarativos gerais (Kowalski, 1974). A lógica computacional também foi bem fortalecida em Edimburgo, graças ao projeto LCF (*Logic for Computable Functions*) (Gordon *et al*., 1979). Esses desenvolvimentos serão examinados mais profundamente nos Capítulos 9 e 12.

Aplicações práticas construídas com lógica de primeira ordem incluem um sistema para avaliar os requisitos de fabricação de produtos eletrônicos (Mannion, 2002), um sistema de raciocínio sobre políticas de acesso a arquivos e gerenciamento de direitos digitais (Halpern e Weissman, 2008), e um sistema para a composição automática de serviços da Web (McIlraith e Zeng, 2001).

Reações à hipótese de Whorf (Whorf, 1956) e o problema da linguagem e do pensamento em geral aparecem em vários livros recentes (Gumperz e Levinson, 1996; Bowerman e Levinson, 2001; Pinker, 2003; Gentner e Goldin-Meadow, 2003). A teoria "teoria" (Gopnik e Glymour, 2002; Tenenbaum *et al*., 2007) vê o aprendizado das crianças sobre o mundo como análoga à construção de teorias científicas. Assim como as previsões de um algoritmo de aprendizagem de máquina dependem fortemente do vocabulário fornecido a ele, a formulação de teorias infantis depende do ambiente linguístico em que a aprendizagem ocorre.

Existem vários textos introdutórios de boa qualidade sobre lógica de primeira ordem incluindo algumas figuras de destaque na história da lógica: Alfred Tarski (1941), Alonzo Church (1956) e W. V. Quine (1982) (que é um dos mais legíveis). Enderton (1972) apresenta uma perspectiva mais orientada para a matemática. Um tratamento altamente formal da lógica de primeira ordem, acompanhado por tópicos muito mais avançados em lógica, é fornecido por Bell e Machover (1977). Manna e Waldinger (1985) apresentam uma introdução interessante à lógica, a partir de uma perspectiva da ciência da computação, assim como Huth e Ryan (2004), que se concentram na verificação de programas. Barwise e Etchemendy (2002) consideram uma abordagem semelhante à utilizada aqui. Smullyan (1995) apresenta resultados de forma concisa, usando o formato de tableau. Gallier (1986) nos oferece uma exposição matemática extremamente rigorosa da lógica de primeira ordem, juntamente com uma grande quantidade de material sobre seu uso em raciocínio automatizado. A obra *Logical Foundations of Artificial Intelligence* (Genesereth e Nilsson, 1987) fornece tanto uma sólida introdução à lógica como apresenta o primeiro tratamento sistemático de agentes lógicos com percepções e ações, e há dois bons manuais: Van Bentham e Ter Meulen (1997) e Robinson e Voronkov (2001). A revista de registro para o campo da lógica matemática é o *Journal of Symbolic Logic*, enquanto *Journal of Applied Logic* ocupa-se com preocupações mais próximas às da inteligência artificial.

EXERCÍCIOS

8.1 Uma base de conhecimento lógico representa o mundo com o uso de um conjunto de sentenças sem estrutura explícita. Por outro lado, uma representação **analógica** tem estrutura física que corresponde diretamente à estrutura do item representado. Considere um mapa rodoviário de nosso país como uma representação analógica de fatos sobre o país — representa fatos em uma linguagem de mapa. A estrutura bidimensional do mapa corresponde à superfície bidimensional da área.

a. Forneça cinco exemplos de *símbolos* na linguagem do mapa.

b. Uma sentença *explícita* é uma sentença que o criador da representação realmente escreve. Uma sentença *implícita* é uma sentença que resulta de sentenças explícitas devido a propriedades da representação analógica. Forneça três exemplos de *sentenças implícitas* e três exemplos de *sentenças explícitas* na linguagem do mapa.

c. Forneça três exemplos de fatos sobre a estrutura física de seu país que não possam ser representados na linguagem do mapa.

d. Forneça dois exemplos de fatos que sejam muito mais fáceis de expressar na linguagem do mapa que em lógica de primeira ordem.

e. Forneça dois outros exemplos de representações analógicas úteis. Quais são as vantagens e as desvantagens de cada uma dessas linguagens?

8.2 Considere uma base de conhecimento que contenha apenas duas sentenças: $P(a)$ e $P(b)$. Essa base de conhecimento tem como consequência lógica $\forall x\ P(x)$? Explique sua resposta em termos de modelos.

8.3 A sentença $\exists x, y\ x = y$ é válida? Explique.

8.4 Escreva uma sentença lógica tal que todo mundo no qual ela é verdadeira contenha exatamente um objeto.

8.5 Considere um vocabulário de símbolos que contenha c símbolos de constantes, p_k símbolos de predicados de cada aridade k e f_k símbolos de funções de cada aridade k, onde $1 \leq k \leq A$. Seja o tamanho do domínio fixo em D. Para qualquer modelo dado, cada símbolo de predicado ou de função é mapeado sobre uma relação ou função, respectivamente, da mesma aridade. Suponha que as funções do modelo permitam que algumas tuplas de entrada não tenham nenhum valor para a função (isto é, o valor é o objeto invisível). Derive uma fórmula que represente o número de modelos possíveis para um domínio com D elementos. Não se preocupe com a eliminação de combinações redundantes.

8.6 Quais das seguintes são sentenças válidas (necessariamente verdadeiras)?

a. $(\exists x\ x = x) \Rightarrow (\forall y\ \exists z\ y = z)$.

b. $\forall x$ P(x) $\vee \neg$P(x).

c. $\forall x$ Smart $(x) \vee (x = x)$.

8.7 Considere uma versão da semântica para lógica de primeira ordem em que são permitidos modelos com domínios vazios. Dê pelo menos dois exemplos de sentenças que são válidas de acordo com a semântica-padrão, mas não de acordo com a nova semântica. Discuta qual resultado dos seus exemplos tem sentido mais intuitivo.

8.8 Será que o fato $\neg$*Cônjuge*(*George, Laura*) resulta do fato de que *Jim* $\neq$ *George* e *Cônjuge*(*Jim, Laura*)? Em caso afirmativo, dê uma prova, se não, forneça axiomas adicionais, conforme necessário. O que acontece se usarmos *Cônjuge* como um símbolo de função unário, em vez de um predicado binário?

8.9 Este exercício usa a função MapColor e os predicados Em(x,y), Fronteira(x,y) e País(x), cujos argumentos são regiões geográficas junto a símbolos constantes para as várias regiões. Em cada um dos seguintes itens expressamos uma sentença e um número de expressões lógicas candidatas. Para cada uma das expressões lógicas, determine se ela (1) expressa corretamente a sentença; (2) é invalida sintaticamente e portanto não tem significado; ou (3) é válida sintaticamente mas não expressa o significado da sentença.

a. Paris e Marseilles localizam-se na França.

(i) *Em*(*Paris* $\wedge$ *Marselha, França*).

(ii) *Em*(*Paris, França*) $\wedge$ *Em*(*Marselha, França*).

(iii) *Em*(*Paris, França*) $\vee$ *Em*(*Marselha, França*).

b. Existe um país que faz fronteira tanto com o Iraque como com o Paquistão.

(i) $\exists\ c\ \ País(c) \land Fronteira(c, Iraque) \land Fronteira(c, Paquistão)$.

(ii) $\exists\ c\ \ País(c) \Rightarrow [Fronteira(c, Iraque) \land Fronteira(c, Paquistão)]$.

(iii) $[\exists\ c\ \ País(c)] \Rightarrow [Fronteira(c, Iraque) \land Fronteira(c, Paquistão)]$.

(iv) $\exists\ c\ \ Fronteiras(País(c), Iraque \land Paquistão)$.

c. Todos os países que fazem fronteira com o Equador estão na América do Sul.

(i) $\forall\ c\ \ País(c) \land Fronteira(c, Equador) \Rightarrow Em(c, América\ do\ Sul)$.

(ii) $\forall\ c\ \ País(c) \Rightarrow [Fronteira(c, Equador) \Rightarrow Em(c, América\ do\ Sul)]$.

(iii) $\forall\ c\ \ [País(c) \Rightarrow Fronteira(c, Equador)] \Rightarrow Em(c, América\ do\ Sul)$.

(iv) $\forall\ c\ \ País(c) \land Fronteira(c, Equador) \land Em(c, América\ do\ Sul)$.

d. Nenhuma região da América do Sul faz fronteira com qualquer região da Europa.

(i) $\neg[\exists\ c, d\ \ Em(c, América\ do\ Sul) \land Em(d, Europa) \land Fronteira(c, d)]$.

(ii) $\forall\ c, d\ \ [Em(c, América\ do\ Sul) \land Em(d, Europa) \Rightarrow \neg Fronteira(c, d)]$.

(iii) $\neg\forall\ c\ \ Em(c, América\ do\ Sul) \Rightarrow \exists\ d\ Em(d, Europa) \land \neg Fronteira(c, d)$.

(iv) $\forall\ c\ \ Em(c, América\ do\ Sul) \Rightarrow \forall d\ Em(d, Europa) \Rightarrow \neg Fronteira(c, d)$.

e. Não existem dois países adjacentes com a mesma cor dentro do mapa.

(i) $\forall\ x, y\ \ \neg País(x) \lor \neg País(y) \lor \neg Fronteira(x, y)] \lor$
$\neg(MapColor(x) = (MapColor(y))$.

(ii) $\forall\ x, y\ \ (País(x) \land País(y) \land Fronteira(x, y) \land \neg(x, y)) \Rightarrow$
$\neg(MapColor(x) = (MapColor(y))$.

(iii) $\forall\ x, y\ \ País(x) \land País(y) \land Fronteira(x, y) \land$
$\neg(MapColor(x) = (MapColor(y))$.

(iv) $\forall\ x, y\ \ (País(x) \land País(y) \land Fronteira(x, y) \Rightarrow MapColor\ (x \neq y)$

8.10 Considere um vocabulário com os símbolos seguintes:

Ocupação (p, o): Predicado. A pessoa p tem a ocupação o.

Cliente $(p1, p2)$: Predicado. A pessoa $p1$ é cliente da pessoa $p2$.

Chefe$(p1, p2)$ Predicado. A pessoa $p1$ é chefe da pessoa $p2$.

Médico, Cirurgião, Advogado, Ator: Constantes que indicam ocupações.

Emília, Joe: constantes que indicam pessoas.

Use esses símbolos para escrever as seguintes asserções em lógica de primeira ordem:

a. Emília é cirurgiã ou advogada.

b. Joe é um ator, mas ele também tem outro trabalho.

c. Todos os cirurgiões são médicos.

d. Joe não tem advogado (ou seja, não é cliente de nenhum advogado).

e. Emília tem um chefe que é advogado.

f. Há um advogado cujos clientes são todos médicos.

g. Todo cirurgião tem um advogado.

8.11 Complete os exercícios a seguir sobre sentenças lógicas:

a. Traduzir em português *bom*, *natural* (sem xs e ys!):

$$\forall\, x, y, FalaIdioma\,(x, l) \land FalaIdioma\,(y, l)$$
$$\Rightarrow Compreende\,(x, y) \land Compreende\,(y, x).$$

b. Explique por que essa sentença é consequência da sentença

$$\forall\, x, y, l\ FalaIdioma\,(x, l) \land FalaIdioma\,(y, l)$$
$$\Rightarrow Compreende\,(x, y).$$

c. Traduza em lógica de primeira ordem as seguintes sentenças:

(i) Compreender leva à amizade.

(ii) A amizade é transitiva.

Lembre-se de definir todos os predicados, funções e constantes que usar.

8.12 Reescreva os dois primeiros axiomas de Peano da Seção 8.3.3 como um único axioma que define *NumNat*(x) de modo a excluir a possibilidade de números naturais, exceto para aqueles gerados pela função sucessor.

8.13 A Equação 8.4 na página 306 define as condições em que um quadrado está com brisa. Aqui consideramos duas outras maneiras de descrever esse aspecto do mundo wumpus.

a. Podemos escrever **regras de diagnóstico** conduzindo de efeitos observados a causas ocultas. Para encontrar os poços, as regras de diagnóstico óbvias informam que, se um quadrado estiver com brisa, algum quadrado adjacente deve conter um poço; e, se um quadrado não estiver com brisa, nenhum quadrado adjacente conterá um poço. Escreva essas duas regras em lógica de primeira ordem e mostre que sua conjunção é logicamente equivalente à Equação 8.4.

b. Podemos escrever **regras causais** que conduzem da causa para o efeito. Uma regra óbvia causal é que um poço faz com que todos os quadrados adjacentes tenham brisa. Escreva essa regra em lógica de primeira ordem, explique por que ela é incompleta em comparação com a Equação 8.4 e forneça o axioma faltante.

8.14 Crie axiomas que descrevam os predicados *NetoOuNeta*, *BisavôOuBisavó*, *Ancestral*, *Irmão*, *Irmã*, *Filha*, *Filho*, *PrimoIrmão*, *Cunhado*, *Cunhada, Tia e Tio.* Descubra a definição adequada de primo em m-ésimo grau n vezes afastado e escreva a definição em lógica de primeira ordem. Agora, anote os fatos básicos representados na árvore genealógica da Figura 8.7. Utilizando um sistema de raciocínio lógico adequado, informe (com TELL) a ele todas as sentenças que você anotou e pergunte (com ASK) ao sistema quem são os netos de Elizabeth, os cunhados de Diana, os bisavós de Zara e os ancestrais de Eugenie.

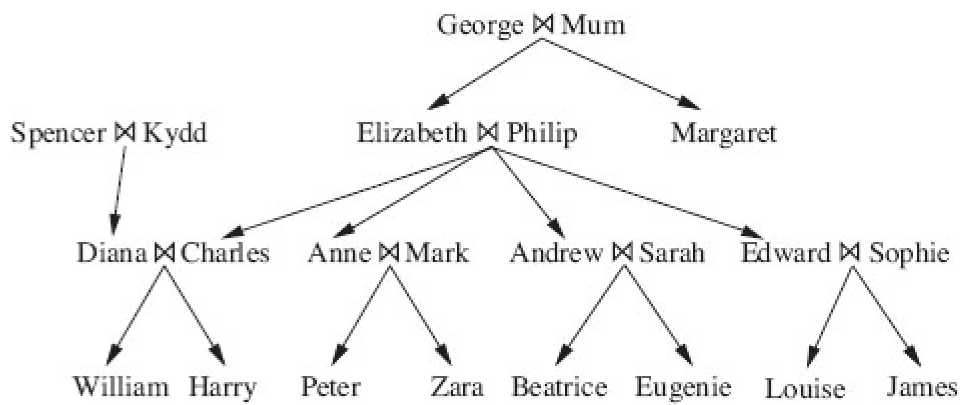


Figura 8.7 Uma árvore genealógica típica. O símbolo "□□" conecta cônjuges e as setas apontam para filhos.

8.15 Explique o que está errado com a seguinte definição proposta do predicado "Pertence a" $\in$:

$\forall\, x, s \;\; x \in \{x|s\}$

$\forall\, x, s \;\; x \in s \;\Rightarrow\; \forall\, y \;\; x \in \{y|s\}$.

8.16 Usando os axiomas de conjuntos como exemplos, escreva axiomas para o domínio de lista, incluindo todas as constantes, funções e predicados mencionados no capítulo.

8.17 Explique o que está errado na definição proposta a seguir de quadrados adjacentes no mundo de wumpus:

$\forall\, x, y\; Adjacente\,([x, y], [x + 1, y]) \wedge Adjacente\,([x, y], [x, y + 1])$.

8.18 Escreva os axiomas necessários para raciocinar sobre a posição do wumpus, usando um símbolo de constante *Wumpus* e um predicado binário *Em*(*Wumpus*, *Posição*). Lembre-se de que só existe um wumpus.

8.19 Assumindo os predicados *PaiOuMãe*(p, θ) e *Feminino*(p) e as constantes *Joan* e *Kevin*, com os significados óbvios, expresse cada uma das seguintes sentenças em lógica de primeira ordem (você pode usar a abreviatura1 $\exists^1$ para significar "existe exatamente um").

a. Joan tem uma filha (possivelmente mais do que uma e, possivelmente, filhos também).

b. Joan tem exatamente uma filha (mas pode ter filhos também).

c. Joan tem exatamente um filho ou filha.

d. Joan e Kevin têm exatamente um filho ou filha juntos.

e. Joan tem pelo menos um filho ou filha com Kevin e não tem filhos com mais ninguém.

8.20 Afirmações aritméticas podem ser escritas em lógica de primeira ordem com o símbolo de predicado <, os símbolos de função + e ×, e os símbolos constantes 0 e 1. Predicados adicionais podem ser também definidos com biconditionais.

a. Represente a propriedade "*x* é um número par".

b. Represente a propriedade "*x* é primo".

c. A conjetura de Goldbach é a conjetura (ainda não provada) de que cada número par é igual à soma de dois primos. Represente essa conjetura como uma sentença lógica.

8.21 No Capítulo 6, foi utilizada igualdade para indicar a relação entre uma variável e seu valor. Por

exemplo, escrevemos *WA* = *vermelho* para significar que a Austrália Ocidental é da cor vermelha. Para representar isso em lógica de primeira ordem, devemos escrever com mais detalhes *CorDe*(*WA*) = *vermelho*. Que inferência incorreta poderia ser estabelecida se escrevêssemos sentenças como *WA* = *vermelho* diretamente como afirmações lógicas?

8.22 Escreva em lógica de primeira ordem a afirmação de que todas as chaves e pelo menos um de cada par de meias serão perdidos eventualmente para sempre usando apenas o seguinte vocabulário: Chave(x), x é uma chave; *Meia*(x), x é uma meia; *Par*(x, y), x e y são um par; *Agora*, o instante atual; *Antes*(t_1, t_2), o instante t_1 vem antes do instante t_2; *Perda*(x, t), o objeto x é perdido no instante t.

8.23 Para cada uma das seguintes sentenças em português, decida se a sentença de primeira ordem que a acompanha é uma boa tradução. Se não, explique por que e a corrija (algumas sentenças podem ter mais de um erro).

a. Não há duas pessoas com o mesmo número do seguro social.

$\neg\exists\ x, y, n\ \ Pessoa\,(x) \land Pessoa\,(x) \Rightarrow [HasSS\,\#\,(x, n) \land HasSS\,\#\,(y, n)].$

b. O número do seguro social do João é igual ao da Maria.

$\exists\ n\ \ HasSS\,\#\,(João, n) \land HasSS\,\#\,(Maria, n).$

C. O número do seguro social de todas as pessoas possui nove dígitos.

$\forall\ x, n\ \ Pessoa\,(x) \Rightarrow [HasSS\,\#\,(x, n) \land Dígitos\,(n, 9)].$

d. Reescreva cada uma das sentenças acima (sem corrigir) usando o símbolo da função SS# em vez do Predicado *HasSS#*.

8.24 Represente as sentenças a seguir em lógica de primeira ordem usando um vocabulário consistente (que você mesmo deve definir):

a. Alguns alunos cursaram francês na primavera de 2001.

b. Todos os alunos que cursam aulas de francês passam.

c. Somente um aluno cursou grego na primavera de 2001.

d. A melhor pontuação em grego é sempre mais alta que a melhor pontuação em francês.

e. Toda pessoa que compra um seguro é inteligente.

f. Ninguém compra um seguro caro.

g. Existe um agente que só vende seguros às pessoas que não têm seguro.

h. Existe um barbeiro que faz a barba de todos os homens na cidade que não fazem a própria barba.

i. Uma pessoa nascida no Reino Unido, que tem como cada um de seus pais um cidadão do Reino Unido ou um residente do Reino Unido, é um cidadão do Reino Unido de nascença.

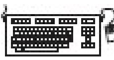
j. Uma pessoa nascida fora do Reino Unido, que tem um de seus pais um cidadão de nascença do Reino Unido, é um cidadão do Reino Unido por descendência.

k. Os políticos podem enganar algumas pessoas todo o tempo, podem enganar todas as pessoas por algum tempo, mas não podem enganar todas as pessoas todo o tempo.

l. Todos os gregos falam a mesma língua. (Use *Fala*(x, l) para dizer que a pessoa x fala o idioma

l.)

8.25 Escreva um conjunto geral de fatos e axiomas para representar a afirmação “Wellington ouviu falar sobre a morte de Napoleão” e responder corretamente à pergunta: “Napoleão ouviu falar sobre a morte de Wellington?”

**8.26** Estenda o vocabulário da Seção 8.4 para definir a adição de números binários de *n* bits. Em seguida, codifique a descrição do somador de quatro bits da Figura 8.8 e formule as consultas necessárias para verificar se de fato ele é correto.

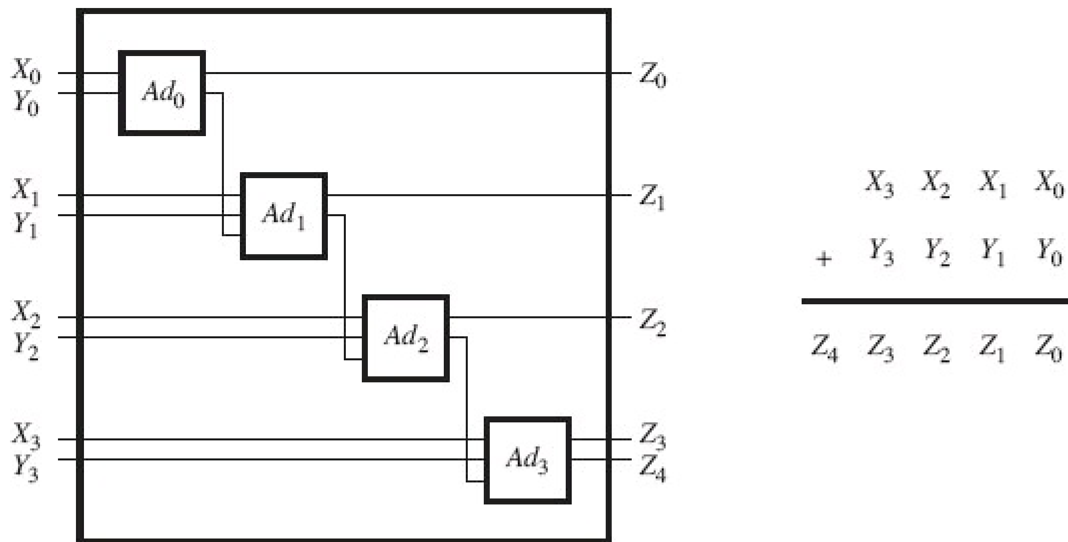


Figura 8.8 Um somador de quatro bits. Cada Ad_i é um somador de um bit, como na Figura 8.6.

8.27 Obtenha uma solicitação de emissão de passaportes para seu país e identifique as regras que definem a elegibilidade para um passaporte, seguindo as etapas delineadas na Seção 8.4.

8.28 Considere uma base de conhecimento de lógica de primeira ordem que descreve o mundo com pessoas, canções, álbuns (ou seja, “*Meet the Beatles*”) e discos (ou seja, instâncias físicas particulares de CDs). O vocabulário contém os símbolos seguintes:

CopiaDe(*d*, *a*): Predicado. O disco *d* é uma cópia do álbum *a*.

Possui(p, *d*): Predicado. A pessoa *p* possui o disco *d*.

Canta(*p*, *s*, *a*): O álbum *a* inclui uma gravação da música *s* cantada pela pessoa *p*.

Escreveu(*p*, *s*): A pessoa *p* escreveu a canção *s*.

McCartney, Gershwin, BHoliday, Joe, EleanorRigby, TheManILove, Revolver:

Constantes com os significados óbvios.

Expresse as seguintes afirmações em lógica de primeira ordem:

a. Gershwin escreveu “*The Man I Love*”.

b. Gershwin não escreveu “Eleanor Rigby”.

c. Ou Gershwin ou McCartney escreveu “*The Man I Love*”.

d. Joe escreveu pelo menos uma canção.

e. Joe possui uma cópia de *Revolver*.

f. Cada canção que McCartney canta em *Revolver* foi escrita por McCartney.

g. Gershwin não escreveu qualquer das músicas de *Revolver*.

h. Cada canção que Gershwin escreveu foi gravada em algum álbum. (Possivelmente canções diferentes foram gravadas em álbuns diferentes.)

i. Há um único álbum que contém todas as músicas que Joe escreveu.

j. Joe possui uma cópia de um álbum em que Billie Holiday canta "*The Man I Love*".

k. Joe possui uma cópia de cada álbum que tem uma canção cantada por McCartney. (É claro que cada álbum diferente é instanciado em um CD físico diferente.)

l. Joe possui uma cópia de todo álbum no qual todas as canções são cantadas por Billie Holiday.

---

[1] Também chamada **cálculo de predicados de primeira ordem** e, às vezes, abreviada como **LPO** ou **CPPO**.

[2] Em contraste, os fatos em **lógica difusa** têm um **grau de verdade** entre 0 e 1. Por exemplo, a sentença "Viena é uma cidade grande" poderia ser verdadeira em nosso mundo apenas até o grau 0,6.

[3] É importante não confundir o grau de crença na teoria da probabilidade com o grau de verdade na lógica difusa. Na realidade, alguns sistemas difusos permitem incerteza (grau de crença) sobre graus de verdade.

[4] Mais à frente, na Seção 8.2.8, examinaremos uma semântica em que cada objeto tem exatamente um nome.

[5] As **expressões** λ oferecem uma notação útil em que novos símbolos de funções são construídos "em tempo de execução". Por exemplo, a função que eleva ao quadrado seu argumento pode ser escrita como $(\lambda x\; x \times x)$ e pode ser aplicada a argumentos da mesma forma que qualquer outro símbolo de função. Uma expressão λ também pode ser definida e usada como um símbolo de predicado (veja o Capítulo 22). O operador lambda em Lisp desempenha exatamente o mesmo papel. Note que o uso de λ desse modo *não* aumenta o poder de expressão formal da lógica de primeira ordem porque qualquer sentença que inclua uma expressão λ pode ser reescrita "unindo-se" seus argumentos para gerar uma sentença equivalente.

[6] Normalmente, seguiremos a convenção de ordenação de argumentos que determina que $P(x, y)$ deve ser interpretada como "x é um P de y".

[7] Existe uma variante do quantificador existencial, em geral escrita como $\exists^1$ ou $\exists!$, que significa "Existe exatamente um". O mesmo significado pode ser expresso com a utilização de declarações de igualdade.

[8] Na verdade, ele tinha quatro, sendo os outros Guilherme e Henrique.

[9] N.R.: Optamos por manter o exemplo original da versão em inglês, assim termos que não têm correspondente direto em português foram traduzidos por nomes com desjunção: PaiOuMãe (Parent) IrmãoOuIrmã (Sibling), FilhoOuFilha (Child), AvôOuAvó (Grandparent) e NetoOuNeta (Grndchild).

[10] Os axiomas de Peano também incluem o princípio de indução, que é uma sentença de lógica de segunda ordem, e não de lógica de primeira ordem. A importância dessa distinção será explicada no Capítulo 9.

[11] De modo semelhante, a maioria das pessoas não identifica cada pássaro que voa sobre sua cabeça à medida que ele migra para regiões mais quentes no inverno. Um ornitologista que deseje estudar padrões de migração, taxas de sobrevivência, e assim por diante, *identificará* cada pássaro, usando um anel em sua perna porque pássaros individuais devem ser monitorados.

CAPÍTULO 9

Inferência em lógica de primeira ordem

Em que definimos procedimentos efetivos para responder a perguntas formuladas em lógica de primeira ordem.

O Capítulo 7 mostrou como a inferência correta e completa pode ser alcançada no caso da lógica proposicional. Neste capítulo, estendemos esses resultados para obter algoritmos capazes de responder a qualquer pergunta enunciada em lógica de primeira ordem que tenha uma resposta possível. A Seção 9.1 introduz regras de inferência para quantificadores e mostra como reduzir a inferência de primeira ordem à inferência proposicional, embora a um custo potencialmente elevado. A Seção 9.2 descreve a ideia de **unificação**, mostrando como ela pode ser usada para construir regras de inferência que funcionam diretamente com sentenças de primeira ordem. Em seguida, descreveremos três importantes famílias de algoritmos de inferência de primeira ordem: o **encadeamento para a frente** e suas aplicações a **bancos de dados dedutivos** e **sistemas de produção** são abordados na Seção 9.3; o **encadeamento para trás** e os sistemas de **programação em lógica** são desenvolvidos na Seção 9.4. Encadeamento para a frente e para trás pode ser muito eficiente, mas são aplicáveis apenas a bases de conhecimento que podem ser expressas como conjuntos de cláusulas de Horn. Sentenças de primeira ordem genéricas requerem prova de teorema baseado em resolução, que é descrito na Seção 9.5.

9.1 INFERÊNCIA PROPOSICIONAL VERSUS INFERÊNCIA DE PRIMEIRA ORDEM

Esta seção e a próxima introduzem as ideias subjacentes aos modernos sistemas de inferência lógica. Começaremos com algumas regras de inferência simples que podem ser aplicadas a sentenças com quantificadores, a fim de obter sentenças sem quantificadores. Essas regras conduzem naturalmente à ideia de que a inferência *de primeira ordem* pode ser realizada convertendo-se a base de conhecimento para a lógica *proposicional* e utilizando-se a inferência *proposicional*, o que já sabemos como fazer. A próxima seção apresenta um atalho óbvio, que conduz a métodos de inferência que manipulam diretamente sentenças de primeira ordem.

9.1.1 Regras de inferência para quantificadores

Vamos começar com quantificadores universais. Suponha que nossa base de conhecimento contenha o folclórico axioma-padrão que afirma que todos os reis ambiciosos são perversos:

$$\forall x\ Rei(x) \land Ambicioso(x) \Rightarrow Perverso(x).$$

Então, parece bastante viável deduzir qualquer das sentenças a seguir:

$$\begin{array}{l} Rei(João) \land Ambicioso(João) \Rightarrow Perverso(João) \\ Rei(Ricardo) \land Ambicioso(Ricardo) \Rightarrow Perverso(Ricardo) \\ Rei(Pai(João)) \land Ambicioso(Pai(João)) \Rightarrow Perverso(Pai(João)). \\ \vdots \end{array}$$

A regra de **instanciação universal** (**IU** para abreviar) afirma que podemos deduzir qualquer sentença obtida pela substituição de um **termo básico** (um termo sem variáveis) para a variável.[1] Para escrever formalmente a regra de inferência, utilizamos a noção de **substituições** introduzida na Seção 8.3. Seja SUBST(θ, a) o resultado da aplicação da substituição θ à sentença a. Então, a regra é escrita como

$$\frac{\forall v\ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}.$$

para qualquer variável v e termo básico g. Por exemplo, as três sentenças dadas anteriormente são obtidas com as substituições $\{x/João\}$, $\{x/Ricardo\}$ e $\{x/Pai(João)\}$.

Na regra de **instanciação existencial**, a variável é substituída por um *novo símbolo de constante* único. A declaração formal é a seguinte: para qualquer sentença a, variável v e símbolo de constante k que não aparece em outro lugar na base de conhecimento,

$$\frac{\exists v\ \alpha}{\text{SUBST}(\{v/k\}, \alpha)}.$$

Por exemplo, da sentença

$$\exists x\ Coroa(x) \land NaCabeça(x, João)$$

podemos deduzir a sentença

$$Coroa(C_1) \land NaCabeça(C_1, João)$$

desde que C_1 não apareça em outro lugar na base de conhecimento. Basicamente, a sentença existencial afirma que existe algum objeto que satisfaz a uma condição, e a aplicação da regra de instanciação existencial está apenas dando um nome a esse objeto. É claro que esse nome não deve pertencer ainda a outro objeto. A matemática oferece um exemplo interessante: vamos supor que descobrimos que existe um número um pouco maior que 2,71828 e que satisfaz a equação $d(x^y)/dy = x^y$ para x. Podemos dar um nome a esse número, como e, mas seria um equívoco dar a ele o nome de

um objeto existente, como p. Em lógica, o novo nome é chamado **constante de Skolem**. A instanciação existencial é um caso especial de um processo mais geral chamado **skolemização**, que cobriremos na Seção 9.5.

Enquanto a instanciação universal pode ser aplicada várias vezes para produzir muitas consequências diferentes, a instanciação do existencial pode ser aplicada uma vez e depois a sentença existencialmente quantificada pode ser descartada. Por exemplo, não precisamos mais da sentença $\exists x\ Matar(x, Vítima)$ uma vez que acrescentamos a sentença $Matar(Assassino, Vítima)$. Em termos estritos, a nova base de conhecimento não é logicamente equivalente à antiga, mas pode se mostrar **inferencialmente equivalente** no sentido de ser satisfatível exatamente quando a base de conhecimento original é satisfatível.

9.1.2 Redução à inferência proposicional

Uma vez que temos regras para deduzir sentenças não quantificadas a partir de sentenças quantificadas, torna-se possível reduzir a inferência de primeira ordem à inferência proposicional. Nesta seção, apresentaremos as ideias principais; os detalhes serão examinados na Seção 9.5.

A primeira ideia é que, da mesma forma que uma sentença existencialmente quantificada pode ser substituída por uma instanciação, uma sentença universalmente quantificada pode ser substituída pelo conjunto de *todas as instanciações possíveis*. Por exemplo, suponha que nossa base de conhecimento contenha apenas as sentenças

$$\begin{aligned}
&\forall x\ Rei(x) \land Ambicioso(x) \Rightarrow Perverso(x)\\
&Rei\ (João)\\
&Ambicioso(João)\\
&Irmão(Ricardo, João).
\end{aligned} \tag{9.1}$$

Em seguida, aplicamos a IU à primeira sentença, usando todas as substituições de termos básicos possíveis a partir do vocabulário da base de conhecimento — nesse caso, $\{x/João\}$ e $\{x/Ricardo\}$. Obtemos

$$\begin{aligned}
&Rei(João) \land Ambicioso(João) \Rightarrow Perverso(João)\\
&Rei(Ricardo) \land Ambicioso(Ricardo) \Rightarrow Perverso(Ricardo),
\end{aligned}$$

e descartamos a sentença universalmente quantificada. Agora, a base de conhecimento é essencialmente proposicional, se visualizarmos as sentenças atômicas básicas — $Rei(João)$, $Ambicioso(João)$, e assim por diante — como símbolos de proposições. Portanto, podemos aplicar qualquer dos algoritmos proposicionais completos do Capítulo 7 para obter conclusões como $Perverso(João)$.

Essa técnica de **proposicionalização** pode se tornar completamente geral, como mostramos na Seção 9.5; ou seja, toda base de conhecimento de primeira ordem e toda consulta podem ser proposicionalizadas de tal modo que a consequência lógica é preservada. Desse modo, temos um procedimento de decisão completo para consequência lógica… ou talvez não. Há um problema: quando a base de conhecimento inclui um símbolo de função, o conjunto de substituições de termos

básicos possíveis é infinito! Por exemplo, se a base de conhecimento menciona o símbolo *Pai*, podem ser construídos infinitamente muitos termos aninhados, como *Pai*(*Pai*(*Pai*(*João*))). Nossos algoritmos proposicionais terão dificuldade com um conjunto de sentenças infinitamente grande.

Felizmente, existe um teorema famoso devido a Jacques Herbrand (1930) afirmando que, se uma sentença é consequência lógica da base de conhecimento de primeira ordem original, então existe uma prova envolvendo apenas um subconjunto *finito* da base de conhecimento proposicionalizada. Tendo em vista que qualquer subconjunto desse tipo tem uma profundidade máxima de aninhamento entre seus termos básicos, podemos encontrar o subconjunto gerando primeiro todas as instanciações com símbolos de constantes (*Ricardo* e *João*), depois todos os termos de profundidade 1 (*Pai*(*Ricardo*) e *Pai*(*João*)), depois todos os termos de profundidade 2, e assim por diante, até sermos capazes de construir uma prova proposicional da sentença que é consequência lógica.

Esboçamos uma abordagem para a inferência de primeira ordem através da proposicionalização que é **completa**, isto é, qualquer sentença que é consequência lógica pode ser provada. Essa é uma realização importante, dado que o espaço de modelos possíveis é infinito. Por outro lado, não sabemos que a sentença *é* consequência lógica até a prova terminar! O que acontece quando a sentença *não* é consequência lógica? Podemos ter conhecimento disso? Para a lógica de primeira ordem, simplesmente não podemos. Nosso procedimento de prova pode prosseguir indefinidamente, gerando termos cada vez mais profundamente aninhados, mas não saberemos se ele ficou paralisado em um loop sem fim ou se a prova está simplesmente prestes a surgir. Isso é muito semelhante ao problema de parada das máquinas de Turing. Alan Turing (1936) e Alonzo Church (1936) provaram, de modos bem diferentes, a inevitabilidade dessa situação. *A questão de consequência lógica no caso da lógica de primeira ordem é* ***semidecidível*** *— isto é, existem algoritmos que respondem "sim" para toda sentença que é consequência lógica, mas não existe nenhum algoritmo que também diga "não" para toda sentença que não é consequência lógica.*

9.2 UNIFICAÇÃO E "ELEVAÇÃO"

A seção precedente descreveu a compreensão da inferência de primeira ordem que existia até o início da década de 1960. O leitor atento (e certamente os lógicos computacionais dos anos 1960) devem ter notado que a abordagem de proposicionalização é bastante ineficiente. Por exemplo, dada a consulta *Perverso*(*x*) e a base de conhecimento da Equação 9.1, parece inconveniente gerar sentenças como *Rei*(*Ricardo*) ∧ *Ambicioso*(*Ricardo*) ⇒ *Perverso*(*Ricardo*). Na verdade, a dedução de *Perverso* (*João*) a partir das sentenças

$$\forall x\ Rei(x) \wedge Ambicioso(x) \Rightarrow Perverso(x)$$
$$Rei(João)$$
$$Ambicioso(João)$$

parece completamente óbvia para um ser humano. Agora, mostraremos como torná-la completamente óbvia para um computador.

9.2.1 Uma regra de inferência de primeira ordem

A inferência de que João é perverso — isto é, que $\{x/João\}$ resolve a consulta *Perverso*(x) — funciona assim: para utilizar a regra de que reis ambiciosos são perversos, encontre algum x tal que x seja um rei e que x seja ambicioso, e depois deduza que esse x é perverso. De modo mais geral, se houver alguma substituição θ que torne cada um dos conjuntos da premissa da implicação idêntica a sentenças que já estão na base de conhecimento, poderemos afirmar a conclusão da implicação depois da aplicação de θ. Nesse caso, a substituição $\theta = \{x/João\}$ alcança esse objetivo. Na realidade, podemos fazer a etapa de inferência realizar um trabalho ainda maior. Vamos supor que, em vez de conhecermos *Ambicioso*(*João*), sabemos que *todo mundo* é ambicioso:

$$\forall y\ Ambicioso(y). \tag{9.2}$$

Então, ainda gostaríamos de poder concluir *Perverso*(*João*), porque sabemos que João é um rei (dado) e que João é ambicioso (porque todo mundo é ambicioso). Para isso funcionar, precisamos encontrar uma substituição, tanto para as variáveis na sentença de implicação quanto para as variáveis nas sentenças que estão na base de conhecimento. Nesse caso, a aplicação da substituição $\{x/João,\ y/João\}$ às premissas da implicação *Rei*(x) e *Ambicioso*(x) e às sentenças da base de conhecimento *Rei*(*João*) e *Ambicioso*(y) as tornará idênticas. Desse modo, podemos deduzir a conclusão da implicação.

Esse processo de inferência pode ser captado como uma única regra de inferência que chamamos ***ModusPonens***[2] **generalizado**: para sentenças atômicas p_i, p_i' e θ, onde existe uma substituição θ tal que SUBST(θ, p_i) = SUBST(θ, p_i'), para todo i,

$$\frac{p_1',\ \ p_2',\ \ \ldots,\ \ p_n',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}\ .$$

Existem $n + 1$ premissas para essa regra: as n sentenças atômicas p_i' e a única implicação. A conclusão é o resultado da aplicação da substituição θ ao consequente θ. Em nosso exemplo:

p_1' é *Rei*(*João*)	p_1 é *Rei*(x)
p_2' é *Ambicioso*(y)	p_2 é *Ambicioso*(x)
θ é $\{x/João, y/João\}$	q é *Perverso*(x)
SUBST(θ, q) é *Perverso*(*João*)	

É fácil mostrar que o *Modus Ponens* generalizado é uma regra de inferência correta. Primeiro observamos que, para qualquer sentença p (cujas variáveis são consideradas universalmente quantificadas) e para qualquer substituição θ,

$$p \models \text{SUBST}(\theta, p)$$

é válida por instanciação universal. Ela é válida em particular para um θ que satisfaz às condições da regra de *Modus Ponens* generalizado. Desse modo, a partir de $p_1', \ldots, p_n'$, podemos deduzir

$$\text{SUBST}(\theta, p_1') \wedge \ldots \wedge \text{SUBST}(\theta, p_n')$$

e, da implicação $p1 \wedge \ldots \wedge pn \Rightarrow \theta$, podemos deduzir

$$\text{SUBST}(\theta, p_1) \wedge \ldots \wedge \text{SUBST}(\theta, p_n) \Rightarrow \text{SUBST}(\theta, q) \,.$$

Agora, θ em *Modus Ponens* generalizado é definido de tal forma que SUBST(θ, pi') = SUBST(θ, pi), para todo i; portanto, a primeira dessas duas sentenças corresponde exatamente à premissa da segunda. Consequentemente, SUBST(θ, θ) segue por *Modus Ponens*.

O *Modus Ponens* generalizado é uma versão **elevada** de *Modus Ponens* — ele eleva o *Modus Ponens* da lógica básica (livre de variáveis) proposicional à lógica de primeira ordem. Veremos no restante do capítulo que é possível desenvolver versões elevadas dos algoritmos de encadeamento para a frente, encadeamento para trás e resolução introduzidos no Capítulo 7. A vantagem fundamental das regras de inferência elevadas sobre a proposicionalização é o fato de elas só efetuarem as substituições necessárias para permitir a derivação de inferências específicas.

9.2.2 Unificação

As regras de inferência elevadas exigem a descoberta de substituições que façam expressões lógicas diferentes parecerem idênticas. Esse processo é chamado **unificação** e é um componente fundamental de todos os algoritmos de inferência de primeira ordem. O algoritmo UNIFICAR recebe duas sentenças e retorna um **unificador** para elas, se existir algum:

UNIFICAR(p, θ) = θ onde SUBST(θ, p) = SUBST(θ, θ)

Vamos examinar alguns exemplos de como UNIFICAR deve se comportar. Suponha que temos uma consulta *AskVars*(*Conhece*(*João*, x)): quem João conhece? Algumas respostas para essa consulta podem ser encontradas descobrindo-se todas as sentenças na base de conhecimento que se unificam com *Conhece*(*João*, x). Aqui estão os resultados da unificação com quatro diferentes sentenças que poderiam estar na base de conhecimento:

$$\text{UNIFICAR}(Conhece(João, x), Conhece(João, Jane)) = \{x/Jane\}$$
$$\text{UNIFICAR}(Conhece(João, x), Conhece(y, Bill)) = \{x/Bill, y/João\}$$
$$\text{UNIFICAR}(Conhece(João, x), Conhece(y, Mãe(y))) = \{y/João, x/Mãe(João)\}$$
$$\text{UNIFICAR}(Conhece(João, x), Conhece(x, Elizabeth)) = falha.$$

A última unificação falha porque x não pode receber os valores *João* e *Elizabeth* ao mesmo tempo. Agora, lembre-se de que *Conhece*(x, *Elizabeth*) significa "Todo mundo conhece Elizabeth" e, assim, *devemos* ser capazes de deduzir que João conhece Elizabeth. O problema só surge porque as duas sentenças utilizam o mesmo nome de variável, x. O problema pode ser evitado **padronizando separadamente** uma das duas sentenças que estão sendo unificadas, o que significa renomear suas variáveis para evitar choques entre nomes. Por exemplo, podemos renomear x em *Conhece*(x, *Elizabeth*) como z_{17} (um novo nome de variável) sem alterar seu significado. Agora, a unificação

funcionará:

$$\text{UNIFICAR}(Conhece(João, x), Conhece(z_{17}, Elizabeth)) = \{x/Elizabeth, x_{17}/João\}.$$

O Exercício 9.12 se aprofunda ainda mais na necessidade de padronizar as variáveis separadamente.

Existe mais uma complicação: dissemos que UNIFICAR deve retornar uma substituição que faça os dois argumentos parecerem iguais. Porém, poderia haver mais de um identificador desse tipo. Por exemplo, UNIFICAR(*Conhece*(*João*, *x*), *Conhece*(*y*, *z*)) poderia retornar {*y*/*João*, *x*/*z*} ou {*y*/*João*, *x*/*João*, *z*/*João*}. O primeiro unificador fornece *Conhece*(*João*, *z*) como resultado da unificação, enquanto o segundo fornece *Conhece*(*João*, *João*). O segundo resultado poderia ser obtido a partir do primeiro por uma substituição adicional {*z*/*João*}; dizemos que o primeiro unificador é *mais geral* que o segundo porque impõe menos restrições sobre os valores das variáveis. Para todo par de expressões que pode ser unificado, existe um único **unificador mais geral** (ou UMG) que é único a menos de renomear e substituir variáveis (por exemplo, {*x*/*João*} e {*y*/*João*} são considerados equivalentes, como também (*x*/*João*, *y*/*João*} e {*x*/*João*, *y*/*x*}.) Nesse caso é {*y*/*João*, *x*/*z*}.

Um algoritmo para calcular unificadores mais gerais encontra-se na Figura 9.1. O processo é muito simples: explorar recursivamente as duas expressões de forma simultânea e "lado a lado", construindo um unificador durante o processo, mas falhando se dois pontos correspondentes nas estruturas não coincidirem. Existe uma única etapa dispendiosa: ao comparar uma variável a um termo complexo, deve-se verificar se a própria variável ocorre dentro do termo; se isso ocorrer, a correspondência falhará porque não será possível construir nenhum unificador consistente. Por exemplo, *S*(*x*) não pode ser unido com *S*(*S*(*x*)). Esse processo, chamado **verificação de ocorrência**, torna a complexidade do algoritmo inteiro quadrática no tamanho das expressões que estão sendo unificadas. Alguns sistemas, inclusive todos os sistemas de programação em lógica, simplesmente omitem a verificação de ocorrência e, como resultado, às vezes fazem inferências incorretas; outros sistemas utilizam algoritmos mais complexos, com complexidade de tempo linear.

```
função UNIFICAR(x, y, q) retorna uma substituição para tornar x e y idênticas
   entradas: x, uma variável, constante, lista ou expressão composta
      y, uma variável, constante, lista ou expressão composta
      q, a substituição construída até agora (opcional, o padrão é vazio)

   se q = falha então retornar falha
   senão se x = y então retornar q
   senão se VARIÁVEL?(x) então retornar UNIFICAR-VAR(x, y, q)
   senão se VARIÁVEL?(y) então retornar UNIFICAR-VAR(y, x, q)
   senão se COMPOSTO?(x) e COMPOSTO?(y) então
      retornar UNIFICAR(ARGS[x], ARGS[y], UNIFICAR(x.OP, y.OP, q))
   senão se LISTA?(x) e LISTA?(y) então
      retornar UNIFICAR(RESTO[x], RESTO[y], UNIFICAR(PRIMEIRO[x], PRIMEIRO[y], q))
   senão retornar falha
```

```
função UNIFICAR-VAR(var, x, q) retorna uma substituição

  se {var/val} ∈q então retornar UNIFICAR(val, x, q)
  senão se {x/val} ∈q então retornar UNIFICAR(var, val, q)
  senão se VERIFICAR-OCORRÊNCIA?(var, x) então retornar falha
  senão retornar adicionar {var/x} a q
```

Figura 9.1 Algoritmo de unificação. O algoritmo funciona comparando as estruturas das entradas, elemento por elemento. A substituição θ, que é o argumento para UNIFICAR, é construída ao longo do processo e é usada para garantir que comparações posteriores serão consistentes com vinculações estabelecidas anteriormente. Em uma expressão composta, como $F(A, B)$, o campo OP seleciona o símbolo de função F, e o campo ARGS escolhe a lista de argumentos (A, B).

9.2.3 Armazenamento e recuperação

Subjacentes às funções TELL e ASK usadas para informar e interrogar uma base de conhecimento, encontram-se as funções mais primitivas ARMAZENAR e RECUPERAR. ARMAZENAR(s) armazena uma sentença s na base de conhecimento e RECUPERAR(θ) retorna todos os unificadores tais que a consulta θ se unifica com alguma sentença na base de conhecimento. O problema que usamos para ilustrar a unificação — encontrar todos os fatos que se unificam com *Conhece*(*João*, *x*) — é um exemplo de RECUPERAR.

O caminho mais simples para implementar ARMAZENAR e RECUPERAR é manter todos os fatos em uma lista longa e unificar cada consulta com cada elemento da lista. Tal processo é ineficiente mas funciona, e é tudo o que você precisa ter para entender o restante do capítulo. O restante desta seção descreve meios de tornar a recuperação mais eficiente e pode ser ignorado em uma primeira leitura.

Podemos tornar RECUPERAR mais eficiente assegurando que só serão experimentadas unificações com sentenças que tenham *alguma* chance de unificação. Por exemplo, não há razão para tentar unificar *Conhece*(*João*, *x*) com *Irmão*(*Ricardo*, *João*). Podemos evitar tais unificações pela **indexação** dos fatos na base de conhecimento.

Um esquema simples chamado **indexação de predicados** coloca todos os fatos de *Conhece* em um único compartimento, e todos os fatos de *Irmão* em outro. Os compartimentos podem ser armazenados em uma tabela de hash para garantir acesso eficiente.

A indexação de predicados é útil quando existem muitos símbolos de predicados, mas apenas algumas cláusulas para cada símbolo. Algumas vezes, no entanto, um predicado tem muitas cláusulas. Por exemplo, suponha que as autoridades fiscais queiram controlar quem emprega quem, utilizando um predicado *Emprega*(*x*, *y*). Esse seria um compartimento muito grande, talvez com milhões de empregadores e dezenas de milhões de empregados. A resposta a uma consulta como *Emprega*(*x*, *Ricardo*) com indexação de predicados exigiria o exame do compartimento inteiro.

Para essa consulta em particular, seria útil se os fatos estivessem indexados por predicado e pelo segundo argumento, talvez com a utilização de uma chave de tabela de hash combinada. Então,

poderíamos simplesmente construir a chave a partir da consulta e recuperar exatamente os fatos que se unificam com a consulta. No caso de outras consultas, como *Emprega*(*IBM*, *y*), precisaríamos ter indexado os fatos combinando o predicado com o primeiro argumento. Por essa razão, os fatos podem ser armazenados sob várias chaves de índice, tornando-se instantaneamente acessíveis a várias consultas com as quais poderiam se unificar.

Dada uma sentença a ser armazenada, é possível construir índices para *todas as consultas possíveis* que se unificam com ela. Para o fato *Emprega*(*IBM*, *Ricardo*), as consultas são:

Emprega(*IBM*, *Ricardo*)	IBM emprega Ricardo?
Emprega(*x*, *Ricardo*)	Quem emprega Ricardo?
Emprega(*IBM*, *y*)	Quem a IBM emprega?
Emprega(*x*, *y*)	Quem emprega quem?

Essas consultas formam um **reticulado de subordinação**, como mostra a Figura 9.2(a). O reticulado tem algumas propriedades interessantes. Por exemplo, o filho de qualquer nó no reticulado é obtido a partir de seu pai por uma única substituição; e o "mais alto" descendente comum de dois nós quaisquer é o resultado da aplicação do unificador mais geral. A porção do reticulado acima de qualquer fato básico pode ser construída sistematicamente (Exercício 9.5). Uma sentença com constantes repetidas tem um reticulado ligeiramente diferente, como mostra a Figura 9.2(b). Símbolos de funções e variáveis nas sentenças a serem armazenadas introduzem estruturas de reticulados ainda mais interessantes.

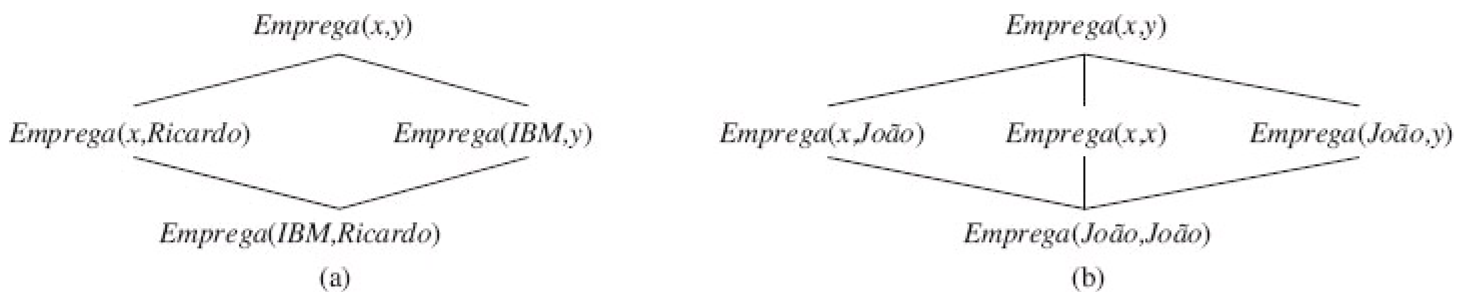


Figura 9.2 (a) Reticulado de subordinação cujo nó mais baixo é a sentença *Emprega*(*IBM*, *Ricardo*). (b) Reticulado de subordinação para a sentença *Emprega*(*João*, *João*).

O esquema que descrevemos funciona muito bem sempre que o reticulado contém um número pequeno de nós. Para um predicado com n argumentos, entretanto, o reticulado contém $O(2^n)$ nós. Se forem permitidos símbolos de funções, o número de nós também será exponencial no tamanho dos termos da sentença a ser armazenada. Isso pode levar a um número enorme de índices. Em certo momento, os benefícios da indexação são superados em valor pelos custos de armazenamento e manutenção de todos os índices. Podemos responder adotando uma política fixa, como manter índices apenas para chaves compostas de um predicado e cada um de seus argumentos ou usando uma política adaptável que crie índices para atender às demandas dos tipos de consultas que estão sendo formuladas. Para a maioria dos sistemas de IA, o número de fatos a serem armazenados é pequeno o bastante para que a indexação eficiente seja considerada um problema resolvido. Para bancos de dados industriais e comerciais, nos quais os fatos quantificam-se na casa dos bilhões, o problema tem sido objeto de estudo intenso e desenvolvimento de tecnologia.

9.3 ENCADEAMENTO PARA A FRENTE

Um algoritmo de encadeamento para a frente para cláusulas definidas proposicionais foi apresentado na Seção 7.5. A ideia é simples: começar com as sentenças atômicas da base de conhecimento e aplicar *Modus Ponens* no sentido para a frente, acrescentando novas sentenças atômicas até não ser mais possível fazer nenhuma inferência adicional. Aqui, explicamos como o algoritmo é aplicado a cláusulas definidas de primeira ordem. Cláusulas definidas como *Situação* ⇒ *Resposta* são especialmente úteis no caso de sistemas que fazem inferências em resposta a informações recém-chegadas. Muitos sistemas podem ser definidos desse modo, e o encadeamento para a frente pode ser implementado muito eficientemente.

9.3.1 Cláusulas definidas de primeira ordem

As cláusulas definidas de primeira ordem são muito semelhantes às cláusulas definidas proposicionais: elas são disjunções de literais dos quais *exatamente um é positivo*. Uma cláusula definida é atômica ou é uma implicação cujo antecedente é uma conjunção de literais positivos e cujo consequente é um único literal positivo. As cláusulas a seguir são cláusulas definidas de primeira ordem:

$$Rei(x) \land Ambicioso(x) \Rightarrow Perverso(x).$$
$$Rei(João).$$
$$Ambicioso(y).$$

Ao contrário dos literais proposicionais, os literais de primeira ordem podem incluir variáveis e, nesse caso, essas variáveis são consideradas universalmente quantificadas (em geral, omitimos os quantificadores universais quando escrevemos cláusulas definidas).

Nem toda base de conhecimento pode ser convertida em um conjunto de cláusulas definidas, devido à restrição de único literal positivo, mas muitas podem. Considere o problema a seguir:

> A lei diz que é crime um americano vender armas a nações hostis. O país Nono, inimigo da América, tem alguns mísseis, e todos foram vendidos pelo Coronel West, um americano.

Provaremos que West é um criminoso. Primeiro, vamos representar esses fatos como cláusulas definidas de primeira ordem. A próxima seção mostrará como o algoritmo de encadeamento para a frente resolve o problema.

"… é crime um americano vender armas a nações hostis":

$$Americano(x) \land Arma(y) \land Vende(x, y, z) \land Hostil(z) \Rightarrow Criminoso(x). \quad (9.3)$$

"Nono… tem alguns mísseis." A sentença $\exists \; x \; Possui(Nono, x) \land Míssil(x)$ é transformada em duas cláusulas definidas por eliminação existencial, introduzindo-se uma nova constante M_1:

$$Possui(Nono, M_1) \quad (9.4)$$

$$Míssil(M_1) \quad (9.5)$$

"Todos foram vendidos pelo Coronel West":

$$Míssil(x) \wedge Possui(Nono, x) \Rightarrow Vende(West, x, Nono) \quad (9.6)$$

Também precisamos saber que mísseis são armas:

$$Míssil(x) \Rightarrow Arma(x) \quad (9.7)$$

e devemos saber que um inimigo da América é considerado "hostil":

$$Inimigo(x, América) \Rightarrow Hostil(x) \quad (9.8)$$

"West, um americano…":

$$Americano(West) \quad (9.9)$$

"O país Nono, inimigo da América…":

$$Inimigo(Nono, América) \quad (9.10)$$

Essa base de conhecimento não contém nenhum símbolo de função e, portanto, é uma instância da classe Datalog de bases de conhecimento. Datalog é uma linguagem restrita a cláusulas definidas de primeira ordem sem símbolos de funções. O nome Datalog deve-se a poder representar o tipo de asserções feitas tipicamente em bancos de dados relacionais. Veremos que a ausência de símbolos de funções torna a inferência muito mais fácil.

9.3.2 Um algoritmo de encadeamento para a frente simples

O primeiro algoritmo de encadeamento para a frente que examinaremos é muito simples, como mostra a Figura 9.3. Começando pelos fatos conhecidos, ele ativa todas as regras cujas premissas são satisfeitas, adicionando suas conclusões aos fatos conhecidos. O processo se repete até a consulta ser respondida (supondo-se que apenas uma resposta seja necessária) ou que nenhum fato novo seja adicionado. Note que um fato não é "novo" se for apenas uma **renomeação** de um fato conhecido. Uma sentença é uma renomeação de outra se elas são sentenças idênticas, exceto pelos nomes das variáveis. Por exemplo, *Gosta*(*x*, *Sorvete*) e *Gosta*(*y*, *sorvete*) são renomeações uma da outra porque diferem apenas na escolha de *x* ou *y*; seus significados são idênticos: todo mundo gosta de sorvete.

função ASK-LPO-EF(*BC*, a) **retorna** uma substituição ou *falso*
  entradas: *BC*, a base de conhecimento, um conjunto de cláusulas definidas de primeira ordem
    a, a consulta, uma sentença atômica
  variáveis locais: *nova*, as novas sentenças deduzidas em cada iteração

```
repita até nova seja vazio
   nova ← { }
   para cada regra em BC faça
      (p1 ∧ … ∧ pn ⇒ q) ← PADRONIZAR-VARIÁVEIS(regra)
      para cada q tal que SUBST(q, p1 ∧ … ∧ pn) = SUBST(q, p′1 ∧ … ∧ p′n)
                  para algum p′1, … p′n em BC
            q′ ← SUBST(q, q)
            se q′ não unifica com alguma sentença já em BC ou nova então faça
            adicionar q′ a nova
            φ ← UNIFICAR(q′, a)
            se φ não é falha então retornar φ
   adicionar nova a BC
retornar falso
```

Figura 9.3 Algoritmo conceitualmente simples, mas muito ineficiente, de encadeamento para a frente. Em cada iteração, ele acrescenta a *BC* todas as sentenças atômicas que podem ser deduzidas em uma única etapa das sentenças de implicação e das sentenças atômicas que já estão em *BC*. A função PADRONIZAR-VARIÁVEIS substitui todas as variáveis em seus argumentos com outras que nunca foram utilizadas antes.

Usaremos nosso problema criminal para ilustrar como funciona ASK-LPO-EF. As sentenças de implicação são 9.3, 9.6, 9.7 e 9.8. Duas iterações são necessárias:

- Na primeira iteração, a regra 9.3 tem premissas não satisfeitas.
  A regra 9.6 é satisfeita com $\{x/M_1\}$, e $Vende(West, M_1, Nono)$ é adicionada.
  A regra 9.7 é satisfeita com $\{x/M_1\}$, e $Arma(M_1)$ é adicionada.
  A regra 9.8 é satisfeita com $\{x/Nono\}$ e $Hostil(Nono)$ é adicionada.
- Na segunda iteração, a regra 9.3 é satisfeita com $\{x/West, y/M_1, z/Nono\}$ e $Criminoso(West)$ é adicionado.

A Figura 9.4 mostra a árvore de prova gerada. Note que nenhuma nova inferência é possível nesse ponto porque toda sentença que poderia ser uma conclusão produzida por encadeamento para a frente já está contida explicitamente na BC. Tal base de conhecimento é chamada **ponto fixo** do processo de inferência. Os pontos fixos alcançados por encadeamento para a frente com cláusulas definidas de primeira ordem são semelhantes aos do encadeamento para a frente proposicional; a principal diferença é que um ponto fixo de primeira ordem pode incluir sentenças atômicas universalmente quantificadas.

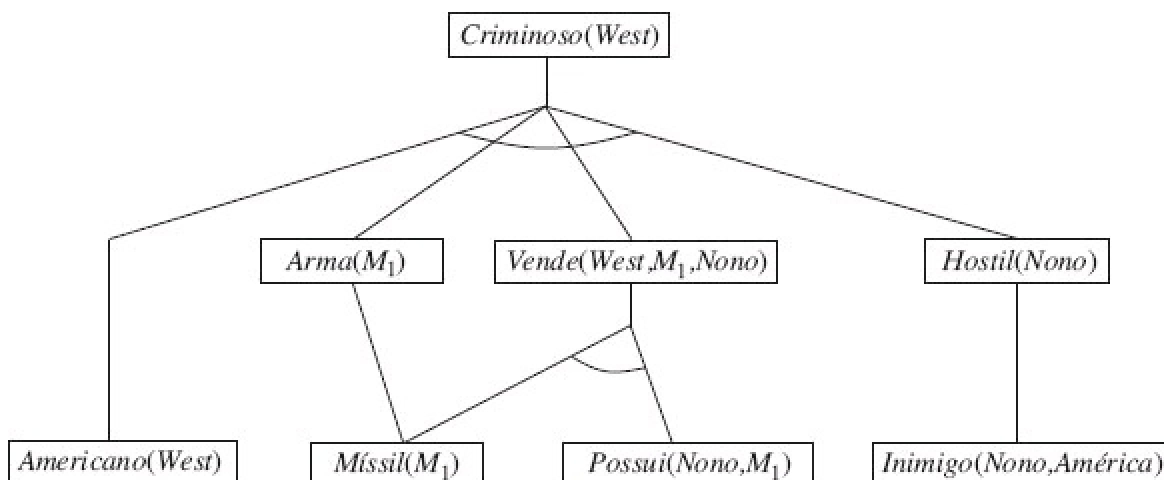


Figura 9.4 A árvore de prova gerada por encadeamento para a frente no exemplo do crime. Os fatos iniciais aparecem no nível inferior, os fatos deduzidos na primeira iteração aparecem no nível intermediário, e os fatos deduzidos na segunda iteração encontram-se no nível superior.

É fácil analisar ASK-LPO-EF. Em primeiro lugar, ele é **correto** porque toda inferência é apenas uma aplicação do *Modus Ponens* generalizado, que é correto. Em segundo lugar, ele é **completo** para bases de conhecimento de cláusulas definidas, ou seja, ele responde a toda consulta cujas respostas são consequências lógicas de qualquer base de conhecimento de cláusulas definidas. No caso de bases de conhecimento Datalog, que não contêm símbolos de funções, a prova de completude é bastante fácil. Começamos efetuando a contagem do número de fatos possíveis que podem ser adicionados, o que determina o número máximo de iterações. Seja k a **aridade** máxima (o número de argumentos) de qualquer predicado, seja p o número de predicados e seja n o número de símbolos de constantes. É claro que não pode haver mais de pn^k fatos básicos distintos; assim, após essa quantidade de iterações, o algoritmo deve ter alcançado um ponto fixo. Então, podemos criar um argumento muito semelhante à prova de completude do encadeamento para a frente proposicional. Os detalhes de como fazer a transição de completude proposicional para compleude de primeira ordem são dados para o algoritmo de resolução na Seção 9.5.

Para cláusulas definidas gerais com símbolos de funções, ASK-LPO-EF pode gerar infinitos novos fatos e, assim, precisamos ser mais cuidadosos. No caso em que uma resposta à sentença da consulta θ é consequência lógica, devemos apelar para o teorema de Herbrand, a fim de estabelecer que o algoritmo encontrará uma prova (veja na Seção 9.5 o caso de resolução). Se a consulta não tivesse nenhuma resposta, o algoritmo poderia não terminar em alguns casos. Por exemplo, se a base de conhecimento incluir os axiomas de Peano

$$NumNat(0)$$
$$\forall n\ NumNat(n) \Rightarrow NumNat(S(n)),$$

o encadeamento para a frente adicionará $NumNat(S(0))$, $NumNat(S(S(0)))$, $NumNat(S(S(S(0))))$, e assim por diante. Em geral, esse problema é inevitável. Como ocorre no caso da lógica de primeira ordem geral, a consequência lógica com cláusulas definidas é semidecidível.

9.3.3 Encadeamento para a frente eficiente

O algoritmo de encadeamento para a frente da Figura 9.3 foi projetado para facilitar a compreensão, e não visando à eficiência de operação. Existem três fontes possíveis de complexidade. Primeiro, o “laço interno” do algoritmo envolve a localização de todos os unificadores possíveis tais que a premissa de uma regra se unifica com um conjunto adequado de fatos na base de conhecimento. Com frequência, esse processo é chamado **correspondência de padrões** e pode ser muito dispendioso. Em segundo lugar, o algoritmo verifica exaustivamente cada regra em toda iteração para ver se suas premissas são satisfeitas, ainda que muito poucas adições sejam feitas à base de conhecimento em cada iteração. Por fim, o algoritmo poderia gerar muitos fatos irrelevantes para o objetivo. Examinaremos cada uma dessas fontes separadamente.

Comparação entre regras e fatos conhecidos

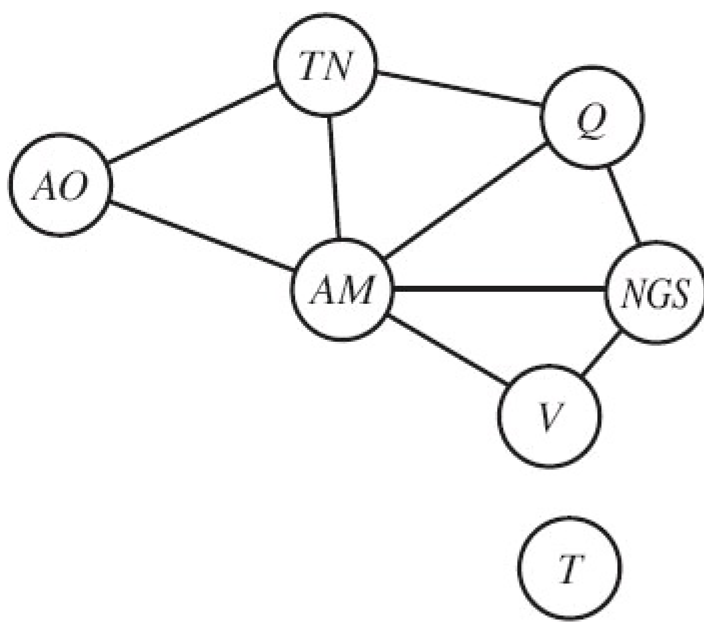
O problema de comparar a premissa de uma regra contra os fatos na base de conhecimento talvez pareça simples. Por exemplo, vamos supor que desejamos aplicar a regra:

$$Míssil(x) \Rightarrow Arma(x).$$

Em seguida, precisamos encontrar todos os fatos que se unificam com $Míssil(x)$; em uma base de conhecimento indexada de modo adequado, isso pode ser feito em tempo constante por fato. Agora, considere uma regra como:

$$Míssil(x) \land Possui(Nono, x) \Rightarrow Vende(West, x, Nono).$$

Novamente, podemos encontrar todos os objetos que Nono possui em tempo constante por objeto; em seguida, para cada objeto, poderíamos verificar se ele é ou não um míssil. Porém, se a base de conhecimento contiver muitos objetos pertencentes a Nono e muito poucos mísseis, será melhor encontrar todos os mísseis primeiro e depois verificar se eles pertencem a Nono. Esse é o problema da **ordenação de elementos da conjunção**: encontrar uma ordenação para resolver os elementos da conjunção da premissa de regra, de tal forma que o custo total seja minimizado. Ocorre que a descoberta da ordenação ótima é NP-difícil, mas existem boas heurísticas disponíveis. Por exemplo, a heurística de **valores restantes mínimos** (VRM) usada para PSRs no Capítulo 6 sugeriria ordenar os elementos da conjunção para procurar primeiro por mísseis, se houvesse menos mísseis que objetos pertencentes a Nono.



Diff (*ao*, *tn*) ∧ *Diff* (*ao*, *am*) ∧
Diff (*tn*, *q*) ∧ *Diff* (*tn*, *am*) ∧
Diff (*q*, *ngs*) ∧ *Diff* (*q*, *am*) ∧
Diff (*ngs*, *v*) ∧ *Diff* (*ngs*, *am*) ∧
Diff (*v*, *am*) ⇒ *Colorable* ( )

Diff (*Vermelho*, *Azul*) *Diff* (*Vermelho*, *Verde*)
Diff (*Verde*, *Vermelho*) *Diff* (*Verde*, *Azul*)
Diff (*Azul*, *Vermelho*) *Diff* (*Azul*, *Verde*)

(b)

Figura 9.5 (a) Grafo de restrição para colorir o mapa da Austrália. (b) PSR de coloração de mapas, expresso como uma única cláusula definida. Cada região do mapa é representada por uma variável cujo valor pode ser uma das constantes *vermelho*, *verde* ou *azul*.

☞ A conexão entre a correspondência de padrões e a satisfação de restrições é na realidade muito estreita. Podemos visualizar cada elemento da conjunção como uma restrição sobre as variáveis que ele contém — por exemplo, *Míssil*(*x*) é uma restrição unária sobre *x*. Estendendo essa ideia, *podemos expressar todo PSR de domínio finito como uma única cláusula definida juntamente com alguns fatos básicos associados*. Considere o problema de coloração de mapa da Figura 6.1, mostrado mais uma vez na Figura 9.5(a). Uma formulação equivalente como uma única cláusula definida é apresentada na Figura 9.5(b). É claro que a conclusão *PodeSerColorido*( ) só poderá ser deduzida se o PSR tiver uma solução. Como os PSRs em geral incluem problemas 3-SAT como casos especiais, podemos concluir que *comparar uma cláusula definida com um conjunto de fatos é NP-difícil*.

Talvez pareça bastante deprimente que o encadeamento para a frente tenha um problema de correspondência NP-difícil em seu loop mais interno. Existem três caminhos para nos alegrar:

- Podemos lembrar que a maioria das regras em base de conhecimento reais é pequena e simples (como as regras em nosso exemplo de crime), em vez de serem regras grandes e complexas (como a formulação de PSR da Figura 9.5). É comum, no mundo de bancos de dados, supor que tanto os tamanhos das regras quanto as aridades de predicados são limitadas por uma constante e se preocupar apenas com a **complexidade de dados**, isto é, a complexidade da inferência como uma função do número de fatos básicos no banco de dados. É fácil mostrar que a complexidade de dados de encadeamento para a frente é polinomial.
- Podemos considerar subclasses de regras para as quais a correspondência é eficiente. Em essência, toda cláusula Datalog pode ser vista como a definição de um PSR e, assim, a correspondência será tratável apenas quando o PSR correspondente for tratável. O Capítulo 6 descreve diversas famílias de PSRs tratáveis. Por exemplo, se o grafo de restrições (o grafo

cujos nós são variáveis e cujos vínculos são restrições) formar uma árvore, o PSR poderá ser resolvido em tempo linear. Exatamente o mesmo resultado é válido no caso da correspondência de regras. Por exemplo, se removermos a Austrália Meridional do mapa da Figura 9.5, a cláusula resultante será

$$Dif(ao, tn) \wedge Dif(tn, q) \wedge Dif(q, ngs) \wedge Dif(ngs, v) \Rightarrow PodeSerColorido(\,)$$

que corresponde ao PSR reduzido mostrado na Figura 6.12. Os algoritmos para resolver PSRs estruturados em árvore podem ser aplicados diretamente ao problema de correspondência de regras.

- Podemos tentar eliminar tentativas redundantes para estabelecer correspondência entre regras no algoritmo de encadeamento para a frente, como descrevemos a seguir.

Encadeamento para a frente incremental

Quando mostramos como funciona o encadeamento para a frente no exemplo do crime, trapaceamos; em particular, omitimos uma parte da correspondência de regras realizada pelo algoritmo da Figura 9.3. Por exemplo, na segunda iteração, a regra

$$Míssil(x) \Rightarrow Arma(x)$$

é comparada a $Míssil(M_1)$ (mais uma vez) e, é claro, a conclusão $Arma(M_1)$ já é conhecida, e portanto nada acontece. Tal correspondência de regras redundante pode ser evitada se fizermos a seguinte observação: *todo fato novo deduzido na iteração t deve ser derivado de pelo menos um fato novo deduzido na iteração* $t - 1$. Isso é verdadeiro porque qualquer inferência que não exigisse um fato novo da iteração $t - 1$ poderia já ter sido realizada na iteração $t - 1$.

Essa observação leva naturalmente a um algoritmo de encadeamento para a frente incremental onde, na iteração t, verificamos uma regra apenas se sua premissa inclui um elemento p_i que se unifica com um fato p'_i recém-deduzido na iteração $t - 1$. A etapa de correspondência de regras fixa p_i para fazê-lo corresponder a p'_i, mas permite que os outros elementos da conjunção da regra correspondam a fatos de qualquer iteração anterior. Esse algoritmo gera exatamente os mesmos fatos em cada iteração que o algoritmo da Figura 9.3, mas é muito mais eficiente.

Com a indexação apropriada, é fácil identificar todas as regras que podem ser ativadas por qualquer fato dado e, na verdade, muitos sistemas reais operam em modo de “atualização”, em que ocorre o encadeamento para a frente em resposta a cada fato novo que seja informado (com TELL) ao sistema. As inferências se propagam em cascata pelo conjunto de regras até ser alcançado o ponto fixo e depois o processo recomeça para o próximo fato novo.

Em geral, apenas uma pequena fração das regras na base de conhecimento é de fato ativada pela inclusão de determinado fato. Isso significa que é realizado muito trabalho redundante na construção repetida de correspondências parciais que têm algumas premissas não satisfeitas. Nosso exemplo de crime é muito pequeno para mostrar isso de forma efetiva, mas note que uma correspondência parcial é construída na primeira iteração entre a regra

$$Americano(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Hostil(z) \Rightarrow Criminoso(x)$$

e o fato *Americano*(*West*). Essa correspondência parcial é então descartada e reconstruída na segunda iteração (quando a regra tem sucesso). Seria melhor reter e completar gradualmente as correspondências parciais à medida que novos fatos chegassem, em vez de descartá-las.

O algoritmo ***rete***[3] foi o primeiro a atacar esse problema. O algoritmo efetua o pré-processamento do conjunto de regras na base de conhecimento para construir uma espécie de rede de fluxo de dados em que cada nó é um literal de uma premissa de regra. As vinculações de variáveis fluem pela rede e são filtradas quando deixam de corresponder a um literal. Se dois literais de uma regra compartilham uma variável — por exemplo, $Vende(x, y, z) \wedge Hostil(z)$ em nosso caso do crime —, as vinculações de cada literal são filtradas através de um nó de igualdade. Uma vinculação de variáveis que alcança um nó para um literal n-ário como $Vende(x, y, z)$ poderia ter de esperar que as vinculações correspondentes às outras variáveis fossem estabelecidas, de modo que o processo pudesse continuar. Em qualquer instante dado, o estado de uma rede *rete* capta todas as correspondências parciais das regras, evitando grande quantidade de repetição de cálculos.

As redes *rete*, bem como diversas melhorias ocorridas a partir delas, constituíram um componente-chave dos chamados **sistemas de produção**, que estavam entre os sistemas de encadeamento para a frente mais antigos de uso difundido.[4] O sistema XCON (originalmente chamado R1, McDermott, 1982) foi construído com o emprego de uma arquitetura de sistema de produção. O XCON continha vários milhares de regras para projetar configurações de componentes de computadores destinados aos clientes da Digital Equipment Corporation. Ele se tornou um dos primeiros sucessos comerciais no campo emergente de sistemas especialistas. Muitos outros sistemas semelhantes foram construídos com a utilização da mesma tecnologia subjacente, que foi implementada na linguagem de uso geral OPS-5.

Os sistemas de produção também são populares em **arquiteturas cognitivas** — ou seja, modelos de raciocínio humano — como ACT (Anderson, 1983) e SOAR (Laird *et al*., 1987). Em tais sistemas, a "memória de trabalho" do sistema tem como modelo a memória humana de curto prazo, e as produções fazem parte da memória de longo prazo. Em cada ciclo de operação, as produções são comparadas à memória de trabalho de fatos. Uma produção cujas condições são satisfeitas pode adicionar ou excluir fatos da memória de trabalho. Em contraste com a situação típica em bancos de dados, com frequência os sistemas de produção têm muitas regras e relativamente poucos fatos. Com uma tecnologia de comparação otimizada de maneira adequada, alguns sistemas modernos podem operar em tempo real com mais de 10 mihões de regras.

Fatos irrelevantes

A última fonte de ineficiência no encadeamento para a frente parece ser intrínseca à abordagem e também surge no contexto proposicional. O encadeamento para a frente faz todas as inferências permissíveis com base nos fatos conhecidos, *mesmo que eles sejam irrelevantes para o objetivo*. Em nosso exemplo de crime, não havia nenhuma regra capaz de tirar conclusões irrelevantes e, assim, a falta de orientação não era um problema sério. Em outros casos (por exemplo, se muitas regras que descrevem os hábitos alimentares dos americanos e os preços dos mísseis), ASK-LPO-EF vai gerar muitas conclusões irrelevantes.

Um modo de se evitar conclusões irrelevantes é usar o encadeamento para trás, como descreve a Seção 9.4. Outra solução é restringir o encadeamento para a frente a um subconjunto selecionado de

regras, como na CONSEQUÊNCIA-LÓGICA-LP-EF?. Uma terceira abordagem emergiu no campo de **bases de dados dedutivas**, que são grandes bases de dados, como bancos de dados relacionais, mas que usam encadeamento para a frente como ferramenta-padrão de inferência em vez de consultas SQL. A ideia é reescrever o conjunto de regras utilizando informações do objetivo, de modo que apenas vinculações de variáveis relevantes — aquelas que pertencem a um conjunto denominado **conjunto mágico** — sejam consideradas durante a inferência para a frente. Por exemplo, se o objetivo é *Criminoso*(*West*), a regra que conclui *Criminoso*(*x*) será reescrita para incluir um elemento extra que limite o valor de *x*:

$$Mágico(x) \land Americano(x) \land Arma(y) \land Vende(x, y, z) \land Hostil(z) \Rightarrow Criminoso(x)$$

O fato *Mágico*(*West*) também é adicionado a BC. Desse modo, mesmo que a base de conhecimento contenha fatos sobre milhões de americanos, apenas o Coronel West será considerado durante o processo de inferência para a frente. O processo completo para definir os conjuntos mágicos e reescrever a base de conhecimento é complexo demais para abordamos aqui, mas a ideia básica é realizar uma espécie de inferência para trás "genérica" a partir do objetivo, a fim de descobrir quais vinculações de variáveis precisam ser limitadas. A abordagem de conjuntos mágicos pode então ser pensada como uma espécie de híbrido entre a inferência para a frente e o pré-processamento para trás.

9.4 ENCADEAMENTO PARA TRÁS

A segunda família importante de algoritmos de inferência lógica utiliza a abordagem de **encadeamento para trás** introduzida na Seção 7.5 para cláusulas definidas. Esses algoritmos funcionam no sentido inverso a partir do objetivo, encadeando regras até encontrar fatos conhecidos que apoiem a prova. Examinaremos o algoritmo básico e depois descreveremos como ele é usado em **programação em lógica**, que é a forma mais amplamente utilizada de raciocínio automatizado. Também veremos que o encadeamento para trás tem algumas desvantagens em comparação com o encadeamento para a frente e estudaremos os meios para superá-las. Finalmente, observaremos a estreita conexão entre a programação em lógica e os problemas de satisfação de restrições.

9.4.1 Um algoritmo de encadeamento para trás

A Figura 9.6 mostra um algoritmo de encadeamento para trás para cláusulas definidas. ASK-LPO-ET (*BC*, *meta*) será provado se a base de conhecimento contiver uma cláusula da forma *le* ⇒ *meta*, onde *le* (lado esquerdo) é uma lista de conjunções. Um fato atômico como *Americano*(*West*) é considerado como uma cláusula cujo *le* é a lista vazia. Agora uma consulta que contém variáveis pode ser provada de várias maneiras. Por exemplo, a consulta *Pessoa*(*x*) poderia ser provada com a substituição {*x*/*João*}, bem como com {*x*/*Ricardo*}. Então implementamos ASK-LPO-ET como um **gerador** — uma função que retorna várias vezes, a cada vez dando um resultado possível.

```
função ASK-LPO-ET(BC, consulta) retorna um gerador de substituições
   retorna OU-LPO-ET(BC, consulta,{ })
```

```
gerador OU-LPO-ET(BC, meta, θ) produz uma substituição
   para cada regra (le ⇒ ld) em RECUPERA-REGRAS-PARA-META(BC, meta) faça
      (le, ld)←PADRONIZAÇÃO-VARIÁVEIS((le, ld))
      para cada q′ em E-LPO-ET(BC, le, UNIFICAR(ld, meta, q) faça
         produzir q′
```

```
gerador E-LPO-ET(BC, metas, q) produz substituição
   se q = falha então retornar
   senão se COMPRIMENTO(metas) = 0 então produz q
   senão faça
      primeiro,resto ← PRIMEIRO(metas), RESTO(metas)
      para cada q′ em OU-LPO-ET (BC, SUBST(q, primeiro), q) faça
         para cada q″ em E-LPO-ET (BC, resto, q′ ) faça
            produzir q″
```

Figura 9.6 Algoritmo de encadeamento para trás simples para bases de conhecimento de primeira ordem.

O encadeamento para trás é uma espécie de busca E/OU — a parte OU porque a consulta objetivo pode ser provada por qualquer regra na base de conhecimento, e a parte E porque todas as conjunções no *le* de uma cláusula devem ser provadas. OU-LPO-ET funciona buscando todas as cláusulas que possam unificar com o objetivo, padronizando as variáveis na cláusula como variáveis novas em folha e, então, se o *ld* (lado direito) da cláusula de fato unificar com o objetivo, provando cada conjunto no *le*, usando E-LPO-ET. Essa função, por sua vez, funciona provando cada um dos conjuntos por vez, guardando a substituição acumulada à medida que avançamos. A Figura 9.7 é a árvore de prova para derivação de *Criminoso*(*West*) a partir das sentenças 9.3 a 9.10.

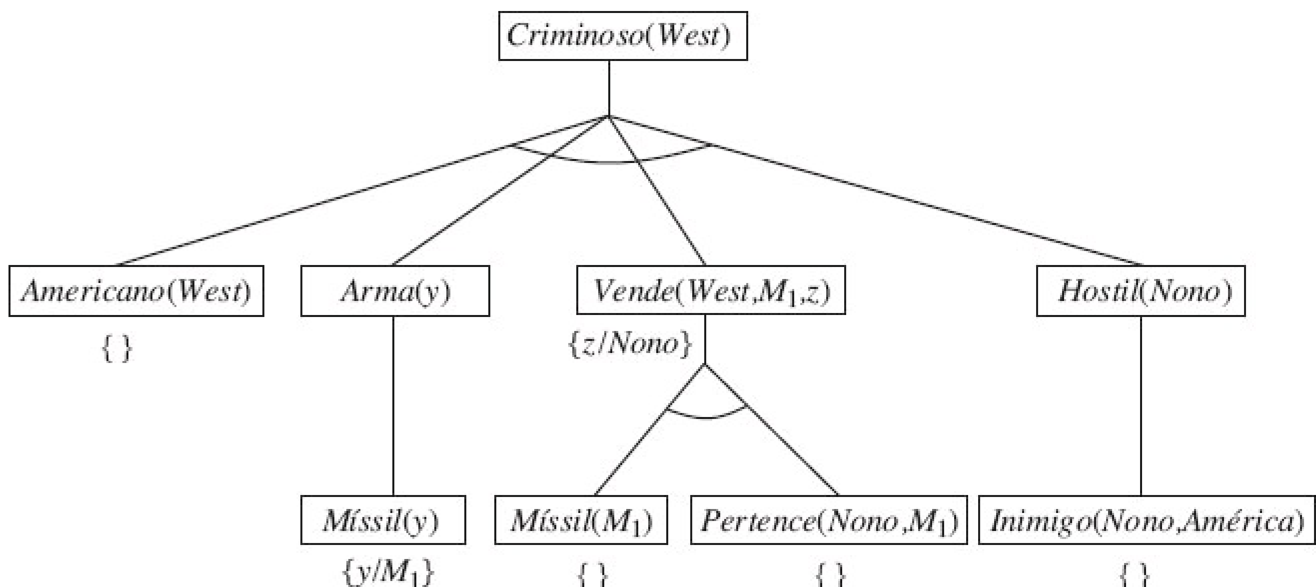


Figura 9.7 Árvore de prova construída por encadeamento para trás para provar que West é um

criminoso. A árvore deve ser lida primeiro na profundidade, da esquerda para a direita. Para provar *Criminoso*(*West*), temos de provar os quatro elementos da conjunção abaixo dele. Alguns desses elementos estão na base de conhecimento e outros exigem encadeamento para trás adicional. As vinculações correspondentes a cada unificação bem-sucedida são mostradas junto ao subobjetivo correspondente. Observe que, uma vez que um subobjetivo em uma conjunção tem sucesso, sua substituição é aplicada aos subobjetivos subsequentes. Desse modo, no momento em que ASK-LPO-ET chegar ao último elemento da conjunção, originalmente *Hostil*(*z*), *z* já estará vinculado a *Nono*.

O encadeamento para trás, como o escrevemos, sem dúvida é um algoritmo de busca em profundidade. Isso significa que seus requisitos de espaço são lineares em relação ao tamanho da prova (negligenciando-se, no momento, o espaço necessário para acumular as soluções). Isso também significa que o encadeamento para trás (diferentemente do encadeamento para a frente) se ressente de problemas com estados repetidos e incompletude. Discutiremos esses problemas e algumas soluções potenciais, mas primeiro veremos como o encadeamento para trás é usado em sistemas de programação em lógica.

9.4.2 Programação em lógica

A programação em lógica é uma tecnologia que se aproxima bastante da incorporação do ideal declarativo descrito no Capítulo 7: os sistemas devem ser construídos expressando-se o conhecimento em linguagem formal, e os problemas devem ser resolvidos executando-se processos de inferência sobre esse conhecimento. Esse ideal é resumido na equação de Robert Kowalski,

$$Algoritmo = Lógica + Controle.$$

O **Prolog** é de longe a mais amplamente utilizada linguagem de programação em lógica. Ele é usado principalmente como linguagem de prototipação rápida e para tarefas de manipulação de símbolos, como a criação de compiladores (Van Roy, 1990) e análise de linguagem natural (Pereira e Warren, 1980). Muitos sistemas especialistas foram escritos em Prolog para domínios jurídicos, médicos, financeiros e outros.

Os programas Prolog são conjuntos de cláusulas definidas escritas em uma notação um pouco diferente da lógica de primeira ordem padrão. Prolog utiliza letras maiúsculas para representar variáveis e letras minúsculas para representar constantes — o oposto da nossa convenção para a lógica. Vírgulas separam elementos de conjunção em uma cláusula, e a cláusula é escrita "ao contrário" do que estamos acostumados: em vez de $A \wedge B \Rightarrow C$, na Prolog temos C: - A, B. Aqui está um exemplo típico:

```
criminoso(X) :- americano(X), arma(Y), vende(X, Y, Z), hostil(Z).
```

A notação [E|L] indica uma lista cujo primeiro elemento é E e o resto é L. Aqui está um programa Prolog para append(X, Y, Z), que tem sucesso se a lista Z é o resultado da concatenação das listas X e Y:

```
append([ ], Y, Y) .
append([A|X], Y, [A|Z]) :- append(X, Y, Z) .
```

Em português, podemos ler essas cláusulas como (1) a concatenação de uma lista vazia com uma lista Y produz a mesma lista Y e (2) [A|Z] é o resultado da concatenação de [A|X] a Y, desde que Z seja o resultado da concatenação de X a Y. Na maioria das linguagens de alto nível podemos escrever uma função recursiva semelhante que descreve como concatenar duas listas. A definição Prolog é realmente muito mais poderosa, no entanto, porque descreve uma *relação* que se mantém entre os três argumentos, em vez de uma função computada a partir de dois argumentos.

Por exemplo, podemos formular a consulta append(X, Y, [1, 2]): quais são as duas listas que podem ser concatenadas para fornecer [1, 2]? Aqui estão as soluções:

```
X=[ ]       Y=[1, 2]
X=[1]       Y=[2]
X=[1, 2]    Y=[ ]
```

A execução de programas Prolog é feita por meio do encadeamento para trás em profundidade, onde as cláusulas são experimentadas na ordem em que são escritas na base de conhecimento. Alguns aspectos de Prolog ficam fora da inferência lógica-padrão:

- Prolog usa a semântica do banco de dados da Seção 8.2.8, em vez da semântica de primeira ordem, e isso é evidente em seu tratamento de igualdade e negação (consulte a Seção 9.4.5).
- Existe um conjunto de funções internas para aritmética. Literais que utilizam esses símbolos de funções são "provados" pela execução de código, em vez da realização de inferência adicional. Por exemplo, o objetivo "X é 4+3" tem sucesso com X associado a 7. Por outro lado, o objetivo "5 é X+Y" falha porque as funções internas não realizam a resolução de equações arbitrárias.[5]
- Existem predicados internos que têm efeitos colaterais quando executados. Entre eles encontramos predicados de entrada/saída e os predicados assert/retract para modificação da base de conhecimento. Tais predicados não têm nenhum equivalente em lógica e podem produzir alguns resultados confusos — por exemplo, se os fatos forem afirmados em uma ramificação da árvore de prova que no final falha.
- A **verificação de ocorrência** é omitida do algoritmo de unificação de Prolog. Isso significa que algumas inferências incorretas podem ser efetuadas; elas quase nunca são um problema na prática.
- Prolog usa encadeamento para trás em profundidade sem verificação de recursão infinita. Isso o torna muito rápido quando dado o conjunto correto de axiomas, mas incompleto quando dado um conjunto errado.

O projeto Prolog representa um compromisso entre declaratividade e eficiência de execução — pelo menos da maneira como a eficiência era entendida na época em que a linguagem Prolog foi projetada.

9.4.3 Implementação eficiente de programas em lógica

A execução de um programa Prolog pode acontecer de dois modos: interpretado e compilado. Em essência, a interpretação consiste em executar o algoritmo ASK-LPO-ET da Figura 9.6, tendo o programa como a base de conhecimento. Dizemos “em essência” porque os interpretadores Prolog contêm uma variedade de melhorias destinadas a maximizar a velocidade. Aqui, vamos considerar apenas duas.

Primeiro, nossa implementação teve que gerenciar explicitamente a iteração sobre os resultados possíveis gerados por cada uma das subfunções. Interpretadores Prolog têm uma estrutura de dados global, uma pilha de **pontos de escolha**, para acompanhar as possibilidades múltiplas de escolha que consideramos em OU-LPO-ET. Essa pilha global é mais eficiente e torna a depuração mais fácil porque o depurador pode se mover para cima e para baixo da pilha.

Em segundo lugar, nossa implementação simples de ASK-LPO-ET gasta um bom tempo gerando e compondo substituições. Em vez de construir substituições explicitamente, a Prolog tem variáveis lógicas que lembram sua vinculação atual. Em qualquer instante, toda variável no programa é não vinculada ou está vinculada a algum valor. Juntos, as variáveis e os valores definem implicitamente a substituição da ramificação atual da prova. A extensão do caminho só pode adicionar novas vinculações de variáveis porque uma tentativa de adicionar uma vinculação diferente a uma variável já vinculada resulta em uma falha de unificação. Quando um caminho na busca falhar, Prolog retornará a um ponto de escolha anterior e então poderá ter de desvincular algumas variáveis. Isso é feito mantendo-se o controle de todas as variáveis que foram vinculadas em uma pilha chamada **trilha**. À medida que cada nova variável é vinculada por UNIFICAR-VAR, a variável é empilhada na trilha. Quando um objetivo falha e chega a hora de retornar a um ponto de escolha anterior, cada uma das variáveis é desvinculada conforme for removida da trilha.

Até mesmo os interpretadores Prolog mais eficientes exigem vários milhares de instruções de máquina por etapa de inferência, devido ao custo do acesso em tabelas de índices, unificação e construção da pilha de chamadas recursivas. Na realidade, o interpretador sempre se comporta como se nunca tivesse visto o programa antes; por exemplo, ele tem de *encontrar* cláusulas que correspondam ao objetivo. Por outro lado, um programa Prolog compilado é um procedimento de inferência para um conjunto específico de cláusulas e, assim, ele *sabe* quais cláusulas correspondem ao objetivo. Basicamente, o Prolog gera um provador de teoremas em miniatura para cada predicado diferente, eliminando grande parte da sobrecarga de interpretação. Também é possível **codificar de modo aberto** a rotina de unificação para cada chamada diferente, evitando desse modo a análise explícita da estrutura de termos (para ver detalhes da unificação em código aberto, consulte Warren *et al*., 1977).

procedimento APPEND(*ax*, *y*, *az*, *continuação*)
  trilha ← APONTADOR-PARA-TRILHA-GLOBAL( )
  se *ax* = [ ] e UNIFICAR(*y*, *az*) **então** CHAMAR(*continuação*)
  RESETAR-TRILHA(*trilha*)
  a,***x***,***z*** ← NOVA-VARIÁVEL( ), NOVA-VARIÁVEL( ), NOVA-VARIÁVEL( )
  se UNIFICAR(*ax*, [*a* | *x*]) e UNIFICAR(*az*, [*a* | *z*]) **então** APPEND(*x*, *y*, *z*, *continuação*)

Figura 9.8 Pseudocódigo que representa o resultado da compilação do predicado Append. A função

NOVA-VARIÁVEL devolve uma nova variável, diferente de todas as outras variáveis usadas até o momento. O procedimento CHAMAR(*continuação*) retoma a execução com a continuação especificada.

Os conjuntos de instruções dos computadores atuais fornecem uma correspondência fraca com a semântica de Prolog e, assim, a maioria dos compiladores Prolog efetua a compilação em uma linguagem intermediária, em vez de fazê-lo diretamente em linguagem de máquina. A linguagem intermediária mais popular é a Warren Abstract Machine, ouWAM, que recebeu esse nome em homenagem a David H. D. Warren, um dos implementadores do primeiro compilador Prolog. A WAM é um conjunto de instruções abstratas adequado para Prolog e que pode ser interpretada ou convertida em linguagem de máquina. Outros compiladores convertem o Prolog em uma linguagem de alto nível, como Lisp ou C, e depois utilizam o compilador dessa linguagem para efetuar a conversão em linguagem de máquina. Por exemplo, a definição do predicado Append pode ser compilada no código mostrado na Figura 9.8. Há diversos detalhes que vale a pena mencionar:

- Em vez de ser necessário pesquisar a base de conhecimento em busca de cláusulas Append, as cláusulas se tornam um procedimento e as inferências são executadas simplesmente chamando-se o procedimento.
- Como descrevemos antes, as vinculações atuais das variáveis são mantidas em uma trilha. A primeira etapa do procedimento salva o estado atual da trilha, de modo que ele pode ser restaurado por RESETAR-TRILHA se a primeira cláusula falhar. Isso vai desfazer quaisquer vinculações geradas pela primeira chamada a UNIFICAR.
- A parte mais complicada é o uso de **continuações** para implementar pontos de escolha. Podemos pensar em uma continuação como a conjugação de um procedimento e uma lista de argumentos que juntos definem o que deve ser feito em seguida, sempre que o objetivo corrente tem sucesso. Simplesmente não seria suficiente retornar de um procedimento como APPEND quando o objetivo tivesse sucesso porque poderia ter sucesso de várias maneiras, e cada uma delas teria de ser explorada. O argumento de continuação resolve esse problema porque pode ser chamado toda vez que o objetivo tem sucesso. No código de APPEND, se o primeiro argumento for vazio e o segundo argumento unifica com o terceiro, o predicado APPEND teve sucesso. Em seguida, chamamos a continuação (com CHAMAR), usando as vinculações apropriadas na trilha, a fim de fazer o que tiver de ser feito em seguida. Por exemplo, se a chamada a APPEND estivesse no nível superior, a continuação imprimiria as vinculações das variáveis.

Antes do trabalho de Warren sobre a compilação de inferência em Prolog, a programação em lógica era lenta demais para uso geral. Os compiladores criados por Warren e outros permitiram que o código Prolog alcançasse velocidades capazes de competir com os de C em diversos benchmarks-padrão (Van Roy, 1990). É claro que o fato de ser possível escrever um planejador ou analisador de linguagem natural em algumas dezenas de linhas de Prolog torna essa linguagem de certa forma mais interessante que C para a prototipação da maioria dos projetos de pesquisa de IA em pequena escala.

A paralelização também pode proporcionar uma aceleração significativa. Existem duas fontes principais de paralelismo. A primeira, chamada **paralelismo-OU**, vem da possibilidade de um objetivo se unificar com muitas cláusulas diferentes na base de conhecimento. Cada uma gera uma

ramificação independente no espaço de busca que pode levar a uma solução potencial, e todas essas ramificações podem ser resolvidas em paralelo. A segunda, chamada **paralelismo-E**, vem da possibilidade de se resolver em paralelo cada elemento da conjunção no corpo de uma implicação. O paralelismo-E é mais difícil de se conseguir porque as soluções para a conjunção inteira exigem vinculações consistentes para todas as variáveis. Cada ramificação conjuntiva deve se comunicar com as outras ramificações para assegurar uma solução global.

9.4.4 Inferência redundante e laços infinitos

Agora vamos examinar o calcanhar de Aquiles de Prolog: a incompatibilidade entre busca em profundidade e árvores de busca que incluem estados repetidos e caminhos infinitos. Considere o programa em lógica a seguir que determina se existe um caminho entre dois pontos em um grafo orientado:

```
caminho(X, Z) :- ligação(X, Z) .
caminho(X, Z) :- caminho(X, Y), ligação(Y, Z).
```

Um grafo simples de três nós, descrito pelos fatos ligação(a, b) e ligação(b, c), é mostrado na Figura 9.9(a). Com esse programa, a consulta caminho(a, c) gera a árvore de prova mostrada na Figura 9.10(a). Por outro lado, se colocarmos as duas cláusulas na ordem

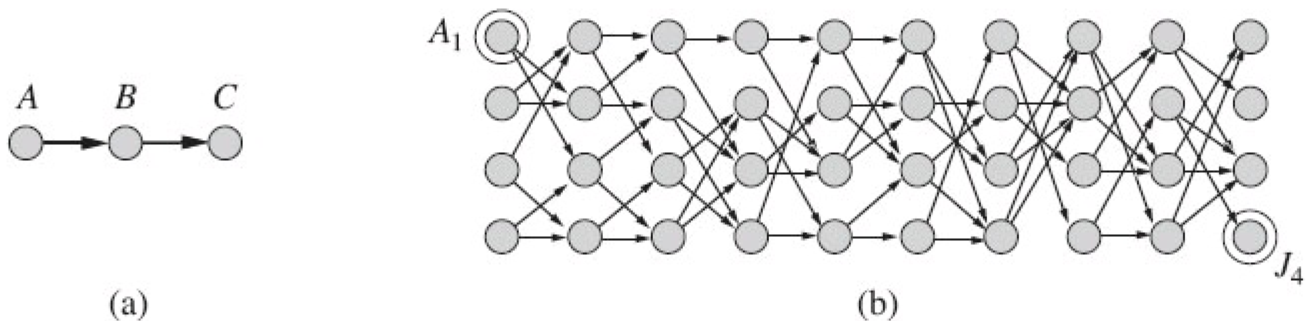


Figura 9.9 (a) Encontrar um caminho de *A* até *C* pode levar o Prolog a um laço infinito. (b) Um grafo em que cada nó está conectado a dois sucessores aleatórios na camada seguinte. Encontrar um caminho de A_1 até J_4 exige 877 inferências.

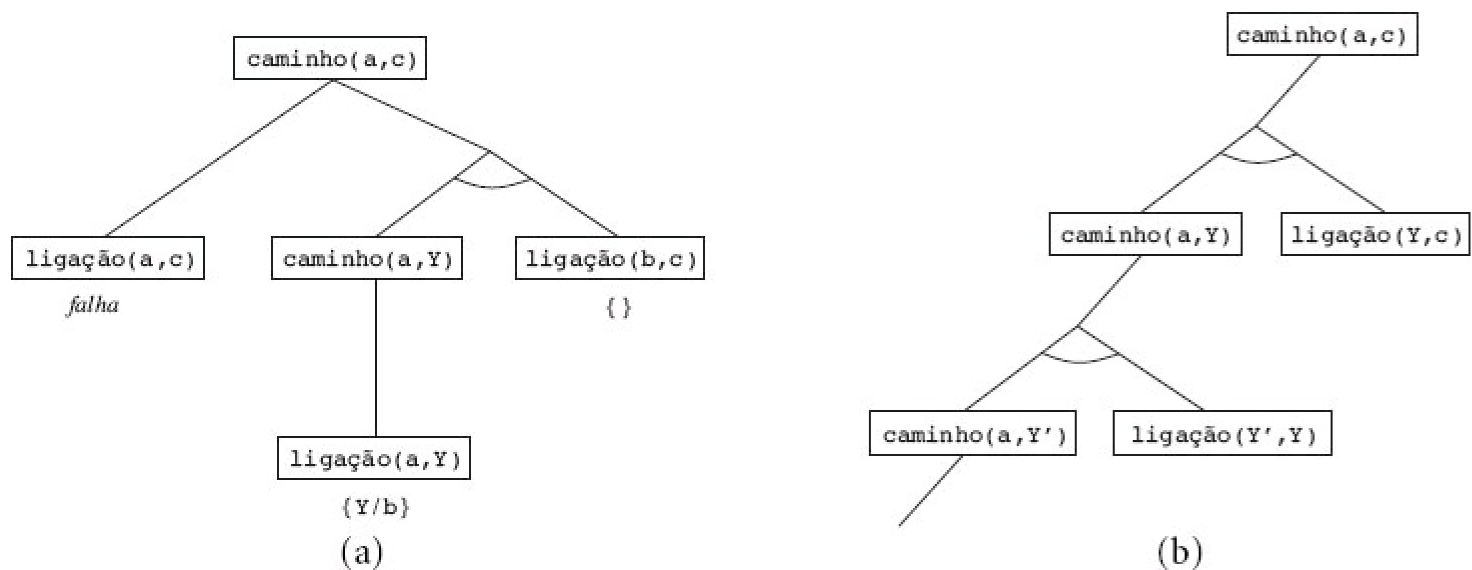


Figura 9.10 (a) Prova de que existe um caminho de *A* até *C*. (b) Árvore de prova infinita gerada quando as cláusulas estão na "ordem errada".

```
caminho(X, Z) :- caminho(X, Y), ligação(Y, Z).
caminho(X, Z) :- ligação(X, Z).
```

o Prolog seguirá o caminho infinito mostrado na Figura 9.10(b). Portanto, Prolog é **incompleto** como provador de teoremas para cláusulas definidas — até mesmo no caso de programas Datalog, como mostra esse exemplo — porque, para algumas bases de conhecimento, ele não consegue provar sentenças que são consequências lógicas. Note que o encadeamento para a frente não se ressente desse problema: uma vez que caminho(a, b), caminho(b, c) e caminho(a, c) são deduzidos, o encadeamento para a frente é suspenso.

O encadeamento para trás em profundidade também tem problemas com computações redundantes. Por exemplo, para encontrar um caminho de A_1 até J_4 na Figura 9.9(b), o Prolog executa 877 inferências, cuja maior parte envolve encontrar todos os caminhos possíveis até os nós a partir dos quais o objetivo é inacessível. Esse problema é semelhante ao problema de estados repetidos descrito no Capítulo 3. A quantidade total de inferências pode ser exponencial em relação ao número de fatos básicos que são gerados. Se, em vez disso, aplicarmos o encadeamento para a frente, no máximo n^2 fatos caminho(X, Y) poderão ser gerados vinculando-se n nós. Para o problema da Figura 9.9(b), são necessárias apenas 62 inferências.

O encadeamento para a frente em problemas de busca de grafos é um exemplo de **programação dinâmica**, no qual as soluções para subproblemas são construídas de forma incremental a partir das soluções de subproblemas menores e são guardadas no cache para evitar a repetição da computação. Podemos obter efeito semelhante em um sistema de encadeamento para trás utilizando a **memoização**, isto é, o processo de guardar no cache soluções para subobjetivos à medida que eles são encontrados e depois reutilizar essas soluções quando o subobjetivo voltar a ocorrer, em vez de repetir a computação anterior. Essa é a abordagem adotada pelos sistemas de **programação em lógica tabulada**, que utilizam mecanismos eficientes de armazenamento e recuperação para executar a memoização. A programação em lógica tabulada combina a orientação para objetivos do encadeamento para trás com a eficiência da programação dinâmica do encadeamento para a frente. Ela também é completa para bases de conhecimento Datalog, o que significa que o programador

precisa se preocupar menos com os laços infinitos. (Ainda é possível obter um laço infinito com predicados como pai(X, Y), que se referem a um número potencial ilimitado de objetos.)

9.4.5 Semântica do banco de dados do Prolog

O Prolog usa a semântica de banco de dados, como discutido na Seção 8.2.8. A suposição de nomes únicos diz que cada constante Prolog e cada termo básico referem-se a um objeto distinto, e o pressuposto de mundo fechado diz que as únicas sentenças que são verdadeiras são aquelas decorrentes da base de conhecimento. Não há como afirmar que uma sentença é falsa em Prolog. Isso torna o Prolog menos expressivo do que a lógica de primeira ordem, mas é parte do que torna o Prolog mais eficiente e mais conciso. Considere as afirmações a seguir em Prolog sobre algumas ofertas de cursos:

$$Curso\ (CS, 101), Curso\ (CS, 102), Curso\ (CS, 106), Curso\ (EE, 101). \tag{9.11}$$

Sob a suposição de nomes únicos, *CS* e *EE* são diferentes (como são 101, 102 e 106), então isso significa que há quatro cursos distintos. Sob a hipótese de mundo fechado não existem outros cursos, por isso há exatamente quatro cursos. Mas, se essas afirmações forem em LPO, em vez de em Prolog, tudo o que poderíamos dizer é que há algo entre um e infinitos cursos. Isso porque as afirmações (em LPO) não negam a possibilidade de que outros cursos não mencionados também sejam oferecidos nem dizem que os cursos mencionados são diferentes uns dos outros. Se quiséssemos traduzir a Equação 9.11 em LPO, obteríamos o seguinte:

$$\begin{aligned} Curso\ (d, n) \Leftrightarrow\ & (d = CS \wedge n = 101) \vee (d = CS \wedge n = 102) \\ & \vee (d = CS \wedge n = 106) \vee (d = EE \wedge n = 101). \end{aligned} \tag{9.12}$$

Isso é chamado **completamento** da Equação 9.11. Ela expressa em LPO a ideia de que existem, no máximo, quatro cursos. Para expressar em LPO a ideia de que existem pelo menos quatro cursos, precisamos escrever o completamento do predicado de igualdade:

$$\begin{aligned} x = y \Leftrightarrow\ & (x = CS \wedge y = CS) \vee (x = EE \wedge y = EE) \vee (x = 101 \wedge y = 101) \\ & \vee (x = 102 \wedge y = 102) \vee (x = 106 \wedge y = 106). \end{aligned}$$

O completamento é útil para a compreensão da semântica de banco de dados, mas, para fins práticos, se seu problema puder ser descrito com a semântica de banco de dados, é mais eficiente raciocinar com Prolog ou algum outro sistema de banco de dados semântico, em vez de traduzir em LPO e raciocinar com um provador de teoremas para LPO completa.

9.4.6 Programação em lógica de restrições

Em nossa discussão sobre o encadeamento para a frente (Seção 9.3), mostramos que os problemas de satisfação de restrições (PSRs) podem ser codificados como cláusulas definidas. A linguagem Prolog padrão resolve esses problemas exatamente do mesmo modo que o algoritmo de retrocesso dado na Figura 6.5.

Tendo em vista que o retrocesso enumera os domínios das variáveis, ele só funciona para PSRs de **domínios finitos**. Em termos de Prolog, deve haver um número finito de soluções para qualquer

objetivo com variáveis não vinculadas. (Por exemplo, o objetivo dif(Q, AM), que informa que Queensland e Austrália Meridional devem ter cores diferentes, tem seis soluções se forem permitidas três cores.) Os PSRs de domínios infinitos — por exemplo, com variáveis de valores inteiros ou reais — exigem algoritmos bem diferentes, como o de propagação de limites ou o de programação linear.

Considere o seguinte exemplo. Definimos o triângulo (X, Y, Z) como um predicado que vale se os três argumentos forem números que satisfazem a desigualdade triangular:

```
triângulo(X, Y, Z) :-
    X>0, Y>0, Z>0, X+Y>=Z, Y+Z>=X, X+Z>=Y.
```

Se formularmos a consulta de Prolog triângulo(3, 4, 5), ela terá sucesso. Por outro lado, se solicitarmos triângulo(3, 4, Z), nenhuma solução será encontrada porque o subobjetivo Z > 0 não poderá ser tratado por Prolog; não podemos comparar um valor desvinculado com 0.

A **programação em lógica de restrições** (PLR) permite que as variáveis sejam *restringidas*, em vez de serem *limitadas*. Uma solução PLR é o conjunto mais específico de restrições sobre as variáveis de consulta que podem ser derivadas da base de conhecimento. Por exemplo, a solução para a consulta triângulo(3, 4, Z) é a restrição 7 >= Z >= 1. Os programas em lógica padrão são apenas um caso especial de PLR em que as restrições da solução devem ser restrições de igualdade, ou seja, vinculações.

Os sistemas de PLR incorporam vários algoritmos de resolução de restrições correspondentes às restrições permitidas na linguagem. Por exemplo, um sistema que permite desigualdades lineares sobre variáveis de valores reais poderia incluir um algoritmo de programação linear para resolver essas restrições. Os sistemas de PLR também adotam uma abordagem muito mais flexível para resolver consultas de programação em lógica padrão. Por exemplo, em vez de retrocesso em profundidade da esquerda para a direita, eles poderiam utilizar qualquer dos algoritmos mais eficientes descritos no Capítulo 6, inclusive a heurística de ordenação de elementos de conjunção, o salto para trás (*backjumping*), a condicionalização de conjuntos de corte, e assim por diante. Portanto, os sistemas de PLR combinam elementos de algoritmos de satisfação de restrições, programação em lógica e bancos de dados dedutivos.

Foram definidos vários sistemas que permitem mais controle ao programador sobre a ordem de busca de inferência. A linguagem MRS (Genesereth e Smith, 1981; Russell, 1985) permite ao programador escrever **metarregras** para determinar que elementos da conjunção serão testados primeiro. O usuário poderia escrever uma regra afirmando que o objetivo com o menor número de variáveis deve ser testado primeiro ou poderia escrever regras específicas de domínio para determinados predicados.

9.5 RESOLUÇÃO

A última de nossas três famílias de sistemas lógicos se baseia na **resolução**. Vimos na Seção 7.5 que a resolução proposicional utilizando refutação é um procedimento de inferência completo para a lógica proposicional. Nesta seção, veremos como estender a resolução à lógica de primeira ordem.

9.5.1 Forma normal conjuntiva para lógica de primeira ordem

Como no caso proposicional, a resolução de primeira ordem exige que as sentenças estejam em **forma normal conjuntiva** (FNC) — ou seja, em uma conjunção de cláusulas, em que cada cláusula é uma disjunção de literais.[6] Os literais podem conter variáveis, que presumimos ser universalmente quantificadas. Por exemplo, a sentença

$$\forall x\ Americano(x) \land Arma(y) \land Vende(x, y, z) \land Hostil(z) \Rightarrow Criminoso(x)$$

torna-se, em FNC,

$$\neg Americano(x) \lor \neg Arma(y) \lor \neg Vende(x, y, z) \lor \neg Hostil(z) \lor Criminoso(x).$$

Toda sentença de lógica de primeira ordem pode ser convertida em uma sentença FNC inferencialmente equivalente. Em particular, a sentença em FNC será não satisfatível exatamente quando a sentença original for não satisfatível; assim, temos uma base para a elaboração de provas por contradição sobre as sentenças em FNC.

O procedimento de conversão para FNC é bem parecido com o caso proposicional, que vimos anteriormente. A principal diferença surge a partir da necessidade de eliminar quantificadores existenciais. Ilustraremos o procedimento convertendo a sentença "Todo mundo que ama todos os animais é amado por alguém" ou

$$\forall x\ [\forall y\ Animal(y) \Rightarrow Ama(x, y)] \Rightarrow [\exists y\ Ama(y, x)]$$

As etapas são:

- **Eliminar implicações:**

$$\forall x\ [\neg\forall y \neg\ Animal(y) \lor Ama(x, y)] \lor [\exists y\ Ama(y, x)]\ .$$

- **Mover ¬ para dentro:** Além das regras habituais para conectivos negados, precisamos de regras para quantificadores negados. Desse modo, temos:

$\neg\forall x\ p$	torna-se	$\exists x\ \neg p$
$\neg\exists x\ p$	torna-se	$\forall x\ \neg p$

  Nossa sentença passa pelas seguintes transformações:

$$\forall x\ [\exists y\ \neg(\neg Animal(y) \lor Ama(x, y))] \lor [\exists y\ Ama(y, x)]\ .$$
$$\forall x\ [\exists y\ \neg\neg Animal(y) \land \neg Ama(x, y)] \lor [\exists y\ Ama(y, x)]\ .$$
$$\forall x\ [\exists y\ Animal(y) \land \neg Ama(x, y)] \lor [\exists y\ Ama(y, x)]\ .$$

  Note que um quantificador universal ($\forall y$) na premissa da implicação se tornou um quantificador existencial. A sentença agora é: "Existe algum animal que x não ama ou (se não for esse o caso) alguém ama x." Sem dúvida, o significado da sentença original foi preservado.

- **Padronizar variáveis:** No caso de sentenças como $(\exists x P(x)) \lor (\exists x\ Q(x))$ que utilizam duas

vezes o mesmo nome de variável, altere o nome de uma das variáveis. Isso evitará confusão mais tarde, quando descartarmos os quantificadores. Desse modo, temos:

$$\forall x\ [\exists y\ Animal(y) \land \neg Ama(x, y)] \lor [\exists\ z\ Ama(z, x)]$$

- **Skolemizar:** A **skolemização** é o processo de remover quantificadores existenciais por eliminação. No caso simples, é semelhante à regra de instanciação do existencial da Seção 9.1: converter $\exists\ x\ P(x)$ em $P(A)$, onde A é uma nova constante. No entanto, não podemos aplicar instanciação existencial para a nossa sentença anterior porque ela não corresponde ao padrão $\exists\ v$ a apenas partes da sentença correspondem ao padrão. Se aplicarmos a regra à cega para as duas partes correspondentes obteremos

$$\forall x\ [Animal(A) \land \neg Ama(x, A)] \lor Ama(B, x)$$

que tem significado completamente errado: ela afirma que todo mundo deixa de amar um animal A específico ou é amado por alguma entidade específica B. De fato, nossa sentença original permite que cada pessoa deixe de amar um animal diferente ou seja amada por uma pessoa diferente. Desse modo, queremos que as entidades de Skolem dependam de x e de z:

$$\forall x\ [Animal(F(x)) \land \neg Ama(x, F(x))] \lor Ama(G(z), x).$$

Aqui F e G são **funções de Skolem**. A regra geral diz que os argumentos da função de Skolem são todas as variáveis universalmente quantificadas, em cujo escopo aparece o quantificador existencial. Como ocorre com a instanciação do existencial, a sentença de Skolem é satisfatível exatamente quando a sentença original é satisfatível.

- **Descartar quantificadores universais:** Nesse momento, todas as variáveis restantes devem ser universalmente quantificadas. Além disso, a sentença é equivalente a outra sentença na qual todos os quantificadores universais são deslocados para a esquerda. Portanto, podemos descartar os quantificadores universais:

$$[Animal(F(x)) \land \neg Ama(x, F(x))] \lor Ama(G(z), x)\ .$$

- **Distribuir $\lor$ sobre $\land$:**

$$[Animal(F(x)) \lor Ama(G(z), x)] \land [\neg Ama(x, F(x)) \lor Ama(G(z), x)]$$

Essa etapa também pode exigir o nivelamento de conjunções e disjunções aninhadas. Agora, a sentença está em FNC e consiste em duas cláusulas. Ela é quase ilegível. (Talvez ajude explicar que a função de Skolem $F(x)$ se refere ao animal potencialmente não amado por x, enquanto $G(z)$ se refere a alguém que poderia amar x.) Felizmente, os seres humanos poucas vezes precisam examinar sentenças FNC — o processo de conversão é automatizado com facilidade.

9.5.2 A regra de inferência de resolução

A regra de resolução para as cláusulas de primeira ordem é simplesmente uma versão elevada da

regra de resolução proposicional dada. Duas cláusulas, que consideramos padronizadas de forma a não compartilharem quaisquer variáveis, podem ser resolvidas se contêm literais complementares. Os literais proposicionais são complementares se um deles é a negação do outro; os literais de primeira ordem são complementares se um deles *se unifica com* a negação do outro. Desse modo, temos:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\textsc{Subst}(\theta, \ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)}$$

onde UNIFICAR(l_1, $\neg m_j$) = θ. Por exemplo, podemos resolver as duas cláusulas

[*Animal*(*F*(*x*)) ∨ *Ama*(*G*(*x*), *x*)] e [¬*Ama*(*u*, *v*) ∨ ¬*Mata*(*u*, *v*)]

eliminando os literais complementares *Ama*(*G*(*x*), *x*) e ¬*Ama*(*u*, *v*), com o unificador $\theta = \{u/G(x), v/x\}$, a fim de produzir a cláusula **resolvente**

[*Animal*(*F*(*x*)) ∨ ¬*Mata*(*G*(*x*), *x*)].

Essa regra é chamada de **resolução binária** porque resolve exatamente dois literais. Sozinha, a regra de resolução binária não gera um procedimento de inferência completo. A regra de resolução completa resolve subconjuntos de literais de cada cláusula que são unificáveis. Uma abordagem alternativa é estender a **fatoração** — a remoção de literais redundantes — ao caso de primeira ordem. A fatoração proposicional reduz dois literais a um só se eles são *idênticos*; a fatoração de primeira ordem reduz dois literais a um só se eles são *unificáveis*. O unificador deve ser aplicado à cláusula inteira. A combinação de resolução binária e fatoração é completa.

9.5.3 Exemplos de provas

A resolução prova que *BC* ⊨ provando que *BC* ∧ ¬a é não satisfatível, isto é, derivando a cláusula vazia. A abordagem algorítmica é idêntica ao caso proposicional descrito na Figura 7.12 e, portanto, não a repetiremos aqui. Em vez disso, forneceremos duas provas como exemplos. A primeira é o exemplo de crime da Seção 9.3. As sentenças em FNC são:

$$\neg Americano(x) \vee \neg Arma(y) \vee \neg Vende(x,y,z) \vee \neg Hostil(z) \vee Criminoso(x)$$
$$\neg Míssil(x) \vee \neg Possui(Nono, x) \vee Vende(West, x, Nono)$$
$$\neg Inimigo(x, América) \vee Hostil(x)$$
$$\neg Míssil(x) \vee Arma(x)$$
$$Possui(Nono, M_1) \qquad Míssil(M_1)$$
$$Americano(West) \qquad Inimigo(Nono, América)$$

Também incluímos o objetivo negado ¬*Criminoso*(*West*). A prova por resolução é mostrada na Figura 9.11. Note a estrutura: a “coluna” única que começa com a cláusula objetivo, utilizando cláusulas da base de conhecimento na resolução até gerar a cláusula vazia. Isso é característico da

resolução sobre bases de conhecimento de cláusulas de Horn. De fato, as cláusulas ao longo da coluna principal correspondem *exatamente* aos valores sucessivos da variável *objetivos* no algoritmo de encadeamento para trás da Figura 9.6. Essa é a razão pela qual sempre optamos pela resolução com uma cláusula cujo literal positivo se unificasse com o literal mais à esquerda da cláusula "atual" da coluna; isso é exatamente o que acontece no encadeamento para trás. Desse modo, o encadeamento para trás é realmente apenas um caso especial de resolução com uma estratégia de controle específica para decidir que resolução executar em seguida.

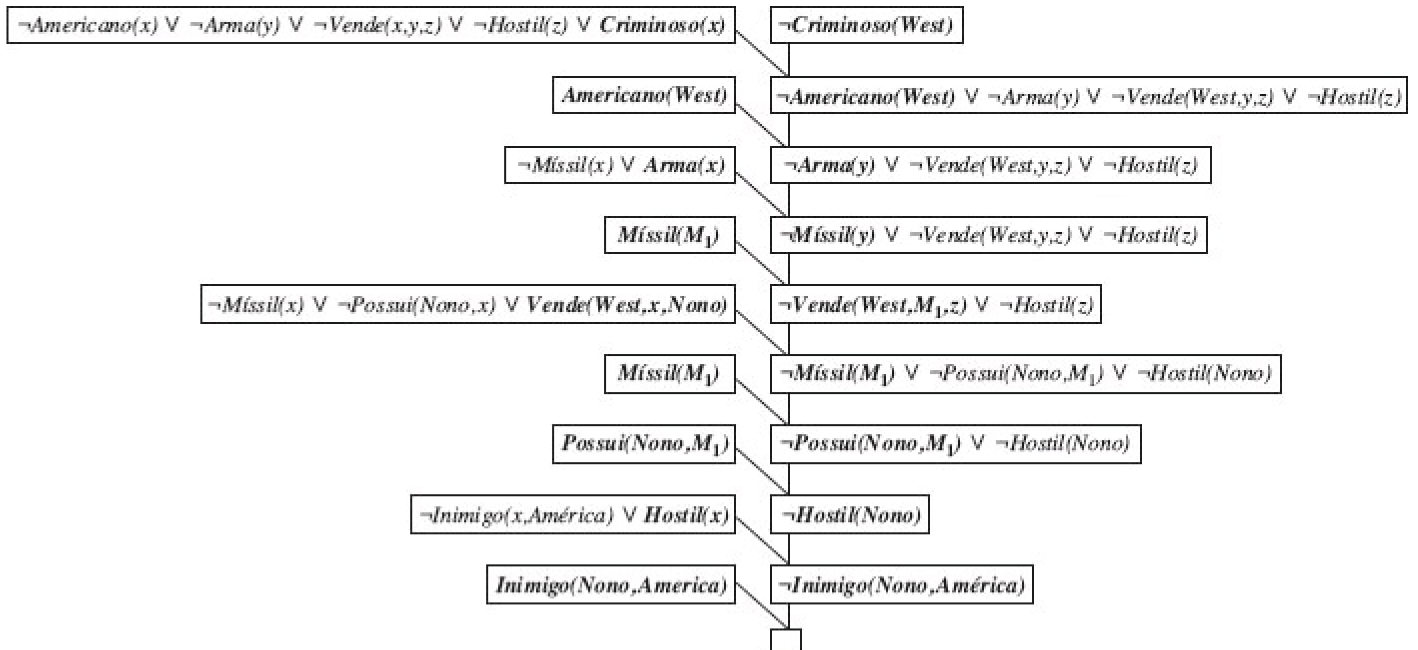


Figura 9.11 Uma prova por resolução de que West é um criminoso. Em cada etapa, os literais que unificam estão em negrito.

Nosso segundo exemplo faz uso da skolemização e envolve cláusulas que não são cláusulas definidas. Isso resulta em uma estrutura de prova um pouco mais complexa. O problema é descrito a seguir em linguagem natural:

Todo mundo que ama todos os animais é amado por alguém.
Qualquer um que mata um animal não é amado por ninguém.
João ama todos os animais.
João ou a Curiosidade matou o gato, que se chama Atum.
A Curiosidade matou o gato?

Primeiro, expressamos as sentenças originais, algum conhecimento prático e o objetivo negado G em lógica de primeira ordem:

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Ama(x, y)] \Rightarrow [\exists y\ Ama(y, x)]$

B. $\forall x\ [\exists z\ Animal(z) \wedge Mata(x, z)] \Rightarrow [\forall y\ \neg Ama(y, x)]$

C. $\forall x\ Animal(x) \Rightarrow Ama(João, x)$

D. $Mata(João, Atum) \vee Mata(Curiosidade, Atum)$

E. $Gato(Atum)$

F. $\forall x\ Gato(x) \Rightarrow Animal(x)$

¬G. ¬*Mata*(*Curiosidade*, *Atum*).

Agora, aplicamos o procedimento de conversão com a finalidade de converter cada sentença para FNC:

A1. *Animal*(*F*(*x*)) ∨ *Ama*(*G*(*x*), *x*)

A2. ¬*Ama*(*x*, *F*(*x*)) ∨ *Ama*(*G*(*x*), *x*)
B. ¬*Ama*(*y, x*) ∨ ¬*Animal*(*z*) ∨ ¬*Ama*(*x*, *z*)
C. ¬*Animal*(*x*) ∨ *Ama*(*João*, *x*)
D. *Mata*(*João*, *Atum*) ∨ *Mata*(*Curiosidade*, *Atum*)
E. *Gato*(*Atum*)
F. ¬*Gato*(*x*) ∨ *Animal*(*x*)
¬G. ¬*Mata*(*Curiosidade*, *Atum*).

A prova por resolução de que a Curiosidade matou o gato é dada na Figura 9.12. Em linguagem natural, a prova poderia ser parafraseada como:

Suponha que a Curiosidade não houvesse matado Atum. Sabemos que João ou a Curiosidade o matou; desse modo, João deve ter matado Atum. Agora, Atum é um gato, e gatos são animais, então Atum é um animal. Tendo em vista que qualquer um que mata um animal não é amado por ninguém, sabemos que ninguém ama João. Por outro lado, João ama todos os animais, então alguém o ama; assim, temos uma contradição. Portanto, a Curiosidade matou o gato.

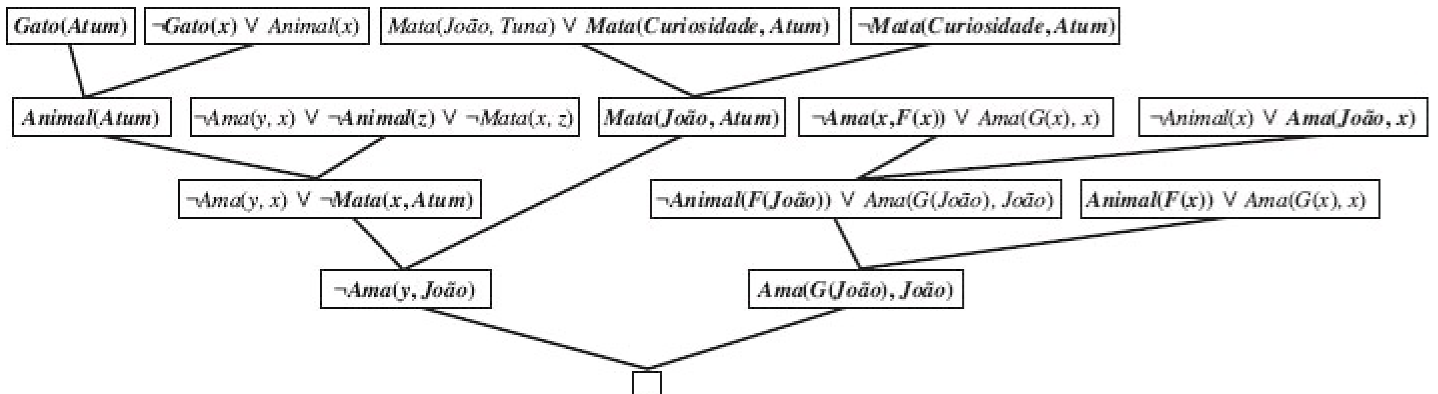


Figura 9.12 Prova por resolução de que a Curiosidade matou o gato. Note o uso da fatoração na derivação da cláusula *Ama*(*G*(*João*),*João*). Observe também, no lado superior direito, que a unificação de *Ama*(*x*, *F*(*x*)) e *Ama*(*João*, *x*) pode apenas ter sucesso depois que as variáveis tiverem sido padronizadas em separado.

A prova responde à pergunta: "A Curiosidade matou o gato?" Porém, frequentemente queremos formular perguntas mais gerais, como: "Quem matou o gato?" A resolução pode fazer isso, mas obter a resposta exige um pouco mais de trabalho. O objetivo é ∃ *w Mata*(*w*, *Atum*) que, quando negada, se torna ¬*Mata*(*w*, *Atum*) em FNC. Repetindo a prova da Figura 9.12 com o novo objetivo negado, obtemos uma árvore de prova semelhante, mas com a substituição {*w*/*Curiosidade*} em uma das etapas. Então, nesse caso, descobrir quem matou o gato é só uma questão de manter o controle das vinculações para as variáveis de consulta da prova.

Infelizmente, a resolução pode produzir **provas não construtivas** para objetivos existenciais. Por exemplo, ¬*Mata*(*w*, *Atum*) se resolve com *Mata*(*João*, *Atum*) ∨ *Mata*(*Curiosidade*, *Atum*) para dar *Mata*(*João*, *Atum*), que se resolve novamente com ¬*Mata*(*w*, *Atum*) para produzir a cláusula vazia. Note que *w* tem duas vinculações diferentes nessa prova; a resolução está nos informando que de fato alguém matou Atum — ou João ou a Curiosidade. Essa não é nenhuma grande surpresa! Uma solução é restringir as etapas de resolução permitidas, de forma que as variáveis da consulta possam ser vinculadas apenas uma vez em determinada prova; então, precisamos ter a possibilidade de efetuar o retrocesso sobre as vinculações possíveis. Outra solução é adicionar um **literal resposta** especial ao objetivo negado, que se torna ¬*Mata*(*w*, *Atum*) ∨ *Resposta*(*w*). Agora, o processo de resolução gera uma resposta sempre que é gerada uma cláusula contendo apenas um *único* literal resposta. Para a prova da Figura 9.12, esse literal resposta é *Resposta*(*Curiosidade*). A prova não construtiva geraria a cláusula *Resposta*(*Curiosidade*) ∨ *Resposta*(*João*), que não constitui de fato uma resposta.

9.5.4 Completude da resolução

Esta seção fornece uma prova de completude da resolução e pode ser seguramente ignorada pelos leitores que estiverem dispostos a aceitá-la com base na fé.

Mostraremos que a resolução é **completa para refutação**, o que significa que, *se* um conjunto de sentenças é não satisfatível, então a resolução sempre será capaz de derivar uma contradição. A resolução não pode ser usada para gerar todas as consequências lógicas de um conjunto de sentenças, mas pode ser usada para estabelecer que dada sentença é consequência lógica do conjunto de sentenças. Consequentemente, ela pode ser utilizada para encontrar todas as respostas para determinada pergunta, $Q(x)$, provando que $BC \land \neg Q(x)$ é insatisfatível.

Consideraremos dado que qualquer sentença em lógica de primeira ordem (sem igualdade) pode ser reescrita como um conjunto de cláusulas em FNC. Isso pode ser provado por indução na forma da sentença, utilizando sentenças atômicas como o caso básico (Davis e Putnam, 1960). Nosso objetivo então é provar o seguinte: *se S é um conjunto de cláusulas não satisfatível*, *então a aplicação de um número finito de passos de resolução a S produzirá uma contradição*.

Nosso esboço de prova segue a prova original apresentada por Robinson, com algumas simplificações feitas por Genesereth e Nilsson (1987). A estrutura básica da prova (Figura 9.13) é a seguinte:

1. Primeiro observamos que, se *S* é não satisfatível, então existe determinado conjunto de *instâncias sem variáveis* das cláusulas de *S* tais que esse conjunto também é não satisfatível (teorema de Herbrand).
2. Em seguida, apelamos para o **teorema básico de resolução** apresentado no Capítulo 7, que declara que a resolução proposicional é completa para sentenças básicas (sem variáveis).
3. Depois, usamos um **lema de elevação** para mostrar que, para qualquer prova por resolução proposicional que utilize o conjunto de sentenças básicas, existe uma prova por resolução de primeira ordem correspondente que utiliza as sentenças de primeira ordem a partir das quais foram obtidas as sentenças básicas.

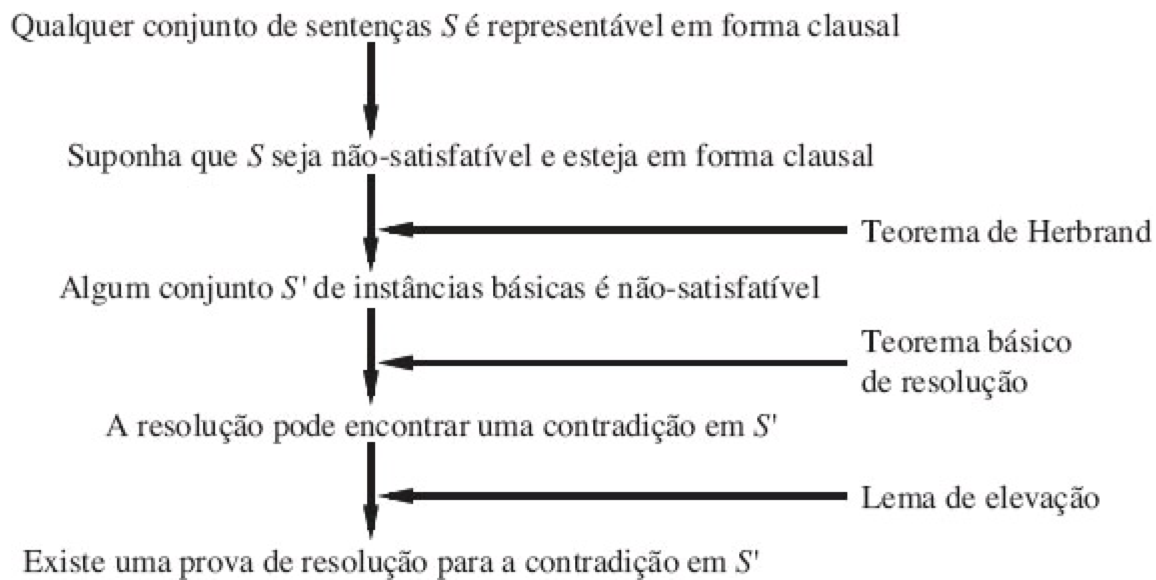


Figura 9.13 Estrutura de uma prova de completude para resolução.

Para executar a primeira etapa, precisaremos de três novos conceitos:

- **Universo de Herbrand:** Se *S* é um conjunto de cláusulas, então H_S, o universo de Herbrand de *S*, é o conjunto de todos os termos básicos (sem variáveis) que podem ser construídos a partir dos seguintes itens:
  a. Os símbolos de funções em *S*, se houver.
  b. Os símbolos de constantes em *S*, se houver; se não houver nenhum, então o símbolo de constante *A*.

  Por exemplo, se *S* contém apenas a cláusula $\neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B)$, então H_S é o conjunto infinito de termos básicos a seguir:

  $$\{A, B, F(A, A), F(A, B), F(B, A), F(B, B), F(A, F(A, A)), \dots\}.$$

- **Saturação:** Se *S* é um conjunto de cláusulas e *P* é um conjunto de termos básicos, então *P*(*S*), a saturação de ***S*** em relação a *P*, é o conjunto de todas as cláusulas básicas obtidas pela aplicação de todas as substituições consistente*s* de termos básicos em *P* com variáveis em *S*.

- **Base de Herbrand:** A saturação de um conjunto *S* de cláusulas em relação a seu universo de Herbrand é chamada base de Herbrand de *S*, representada como $H_S(S)$. Por exemplo, se *S* contém somente a cláusula que acabamos de apresentar, então $H_S(S)$ é o conjunto infinito de cláusulas

$$\{\neg P(A, F(A, A)) \vee \neg Q(A, A) \vee R(A, B),$$
$$\neg P(B, F(B, A)) \vee \neg Q(B, A) \vee R(B, B),$$
$$\neg P(F(A, A), F(F(A, A), A)) \vee \neg Q(F(A, A), A) \vee R(F(A, A), B),$$
$$\neg P(F(A, B), F(F(A, B), A)) \vee \neg Q(F(A, B), A) \vee R(F(A, B), B) \dots\}$$

Essas definições nos permitem enunciar uma forma do **teorema de Herbrand** (Herbrand, 1930):

Se um conjunto *S* de cláusulas é não satisfatível, então existe um subconjunto finito de $H_S(S)$ que

também é não satisfatível.

Seja S' esse subconjunto finito de sentenças básicas. Agora, podemos apelar para o teorema básico de resolução para mostrar que o **fecho da resolução** $FR(S')$ contém a cláusula vazia. Isto é, a execução da resolução proposicional até o completamento sobre S' derivará uma contradição.

Agora que estabelecemos que sempre existe uma prova por resolução envolvendo algum subconjunto finito da base de Herbrand de *S*, o próximo passo é mostrar que existe uma prova por resolução que utiliza as cláusulas do próprio *S*, que não são necessariamente cláusulas básicas. Começamos considerando uma única aplicação da regra de resolução. Robinson enunciou o seguinte:

> Sejam C_1 e C_2 duas cláusulas sem variáveis compartilhadas, e sejam C'_1 e C'_2 instâncias básicas de C_1 e C_2. Se C' é um resolvente de C'_1 e C'_2, então existe uma cláusula *C* tal que (1) *C* é um resolvente de C_1 e C_2, e (2) C' é uma instância básica de *C*.

TEOREMA DA INCOMPLETUDE DE GÖDEL

Estendendo um pouco a linguagem da lógica de primeira ordem para permitir o uso do **esquema de indução matemática** em aritmética, Gödel conseguiu mostrar, em seu **teorema de incompletude,** que existem sentenças aritméticas verdadeiras que não podem ser provadas.

A prova do teorema da incompletude está um pouco além do escopo deste livro, ocupando, como de fato ocupa, pelo menos 30 páginas; porém, podemos apresentar aqui uma ideia da prova. Começamos com a teoria lógica dos números. Nessa teoria, existe uma única constante, 0, e uma única função, *S* (a função sucessora). No modelo pretendido, $S(0)$ denota 1, $S(S(0))$ denota 2, e assim por diante; portanto, a linguagem tem nomes correspondentes a todos os números naturais. O vocabulário também inclui os símbolos de funções +, × e *Exp* (exponenciação), além do conjunto habitual de conectivos e quantificadores lógicos. A primeira etapa é notar que o conjunto de sentenças que podemos escrever nessa linguagem pode ser enumerado. (Imagine definir uma ordem alfabética sobre os símbolos e depois organizar em ordem alfabética cada um dos conjuntos de sentenças de comprimento 1, 2, e assim por diante.) Podemos então numerar cada sentença α com um número natural único #α (o **número de Gödel**). Isso é crucial: a teoria dos números contém um nome para cada uma de suas sentenças. De modo semelhante, podemos numerar cada prova possível *P* com um número de Gödel $G(P)$ porque uma prova é simplesmente uma sequência finita de sentenças.

Agora, vamos supor que tenhamos um conjunto recursivamente enumerável *A* de sentenças que são declarações verdadeiras sobre os números naturais. Lembrando que *A* pode ser identificado por um dado conjunto de inteiros, podemos imaginar a escrita em nosssa linguagem de uma sentença $\alpha(j, A)$ do seguinte tipo:

> $\forall i$ *i* não é o número de Gödel de uma prova da sentença cujo número de Gödel é *j*, onde a prova utiliza apenas premissas existentes em *A*.

Então, seja σ a sentença $\alpha(\#\sigma, A)$, ou seja, uma sentença que enuncia sua própria não demonstrabilidade a partir de *A* (o fato de que essa sentença sempre existe é verdade, mas não

completamente óbvio).

Agora, criamos o seguinte argumento engenhoso: suponha que σ possa ser provada a partir de *A*; então, σ é falsa (porque σ afirma que ela própria não pode ser provada). Mas então temos uma sentença falsa que é demonstrável a partir de *A* e, assim, *A* não pode consistir apenas em sentença verdadeiras — uma violação de nossa premissa. Por conseguinte, σ *não* é demonstrável a partir de *A*. No entanto, isso é exatamente o que a própria σ afirma; consequentemente, σ é uma sentença verdadeira.

Desse modo, mostramos (economizando 29 páginas e meia) que, para qualquer conjunto de sentenças verdadeiras da teoria dos números e, em particular, para qualquer conjunto de axiomas básicos, existem outras sentenças verdadeiras que *não* podem ser provadas a partir desses axiomas. Isso estabelece, entre outras coisas, que nunca podemos provar todos os teoremas de matemática *dentro de qualquer sistema de axiomas dado.* Sem dúvida, essa foi uma descoberta importante para a matemática. Seu significado para a IA foi amplamente debatido, a partir de especulações feitas pelo próprio Gödel. Estudaremos o debate no Capítulo 26.

Esse lema é chamado **lema de elevação** porque eleva uma etapa de prova de cláusulas básicas até cláusulas gerais de primeira ordem. Para provar esse lema básico de elevação, Robinson teve de criar a unificação e derivar todas as propriedades de unificadores mais gerais. Em vez de repetir a prova aqui, vamos simplesmente ilustrar o lema:

$$\begin{aligned}
C_1 &= \neg P(x, F(x, A)) \lor \neg Q(x, A) \lor R(x, B)\\
C_2 &= \neg N(G(y), z) \lor P(H(y), z)\\
C'_1 &= \neg P(H(B), F(H(B), A)) \lor \neg Q(H(B), A) \lor R(H(B), B)\\
C'_2 &= \neg N(G(B), F(H(B), A)) \lor P(H(B), F(H(B), A))\\
C' &= \neg N(G(B), F(H(B), A)) \lor \neg Q(H(B), A) \lor R(H(B), B)\\
C &= \neg N(G(y), F(H(y), A)) \lor \neg Q(H(y), A) \lor R(H(y), B).
\end{aligned}$$

Vemos que de fato C' é uma instância básica de C. Em geral, para C'_1 e C'_2 terem quaisquer resolventes, elas devem ser construídas pela aplicação inicial a C_1 e C_2 do unificador mais geral de um par de literais complementares em C_1 e C_2. Do teorema de elevação, é fácil derivar um enunciado semelhante sobre qualquer sequência de aplicações da regra de resolução:

> Para qualquer cláusula C' no fecho da resolução de S' existe uma cláusula C no fecho da resolução de S, tal que C' é uma instância básica de C e a derivação de C tem o mesmo comprimento que a derivação de C'.

A partir desse fato, segue-se que, se a cláusula vazia aparecer no fecho da resolução de S', ela também deverá aparecer no fecho da resolução de S. Isso ocorre porque a cláusula vazia não pode ser uma instância básica de qualquer outra cláusula. Para recapitular: mostramos que, se S é não satisfatível, então existe uma derivação finita da cláusula vazia que utiliza a regra de resolução.

A elevação da demonstração de teoremas das cláusulas básicas para as cláusulas de primeira ordem proporciona grande aumento de potencial. Esse aumento provém do fato de que a prova de

primeira ordem só precisa instanciar variáveis até onde for necessário para a prova, enquanto os métodos de cláusulas básicas eram obrigados a examinar enorme número de instanciações arbitrárias.

9.5.5 Igualdade

Nenhum dos métodos de inferência descritos até agora neste capítulo trata uma asserção da forma $x = y$. Existem três abordagens distintas que podem ser adotadas. A primeira abordagem é axiomatizar a igualdade — escrever sentenças sobre a relação de igualdade na base de conhecimento. Precisamos afirmar que a igualdade é reflexiva, simétrica e transitiva, e também temos de afirmar que podemos substituir itens iguais por itens iguais em qualquer predicado ou função. Assim, precisamos de três axiomas básicos e depois de um axioma para cada predicado e função:

$$\begin{array}{l}
\forall x\ x = x \\
\forall x, y\ x = y \Rightarrow y = x \\
\forall x, y, z\ x = y \wedge y = z \Rightarrow x = z \\
\\
\forall x, y\ x = y \Rightarrow (P_1(x) \Leftrightarrow P_1(y)) \\
\forall x, y\ x = y \Rightarrow (P_2(x) \Leftrightarrow P_2(y)) \\
\quad \vdots \\
\forall w, x, y, z\ w = y \wedge x = z \Rightarrow (F_1(w, x) = F_1(y, z)) \\
\forall w, x, y, z\ w = y \wedge x = z \Rightarrow (F_2(w, x) = F_2(y, z)) \\
\quad \vdots
\end{array}$$

Dadas essas sentenças, um procedimento de inferência padrão como a resolução pode executar tarefas que exigem raciocínio sobre a igualdade, como a resolução de equações matemáticas. No entanto, esses axiomas irão gerar uma porção de conclusões, e a maioria delas não é útil para uma prova. Então, houve uma busca de formas mais eficientes de lidar com a igualdade. Uma alternativa foi adicionar regras de inferência em vez de axiomas. A regra mais simples, a **demodulação**, toma uma cláusula unitária $x = y$ e alguma cláusula a que contém o termo x e produz uma nova cláusula formada pela substituição de y por x dentro de a. Funciona se o termo dentro de a unifica com x; não precisa ser exatamente igual a x.

Observe que a demodulação é direcional; dado $x = y$, o x sempre é substituído por y, nunca o contrário. Isso significa que a demodulação pode ser usada para simplificar expressões usando demoduladores, como $x + 0 = x$ ou $x^1 = x$. Como outro exemplo, dado

Pai(*Pai*(*x*)) = *AvoPaterno*(*x*)
Data de nascimento(*Pai*(*Pai*(*Bella*)), 1926)

podemos concluir pela demodulação

Data de nascimento(*AvoPaterno*(*Bella*), 1926).

Mais formalmente, temos:

- **Demodulação:** Para quaisquer termos x, y e z, onde z aparece em algum lugar no literal m_i e onde UNIFICAR(x, z) = θ,

$$\frac{x = y, \qquad\qquad m_1 \vee \cdots \vee m_n}{\text{SUB}(\text{SUBST}(\theta, x), \text{SUBST}(\theta, y), m_1 \vee \cdots \vee m_n)} .$$

onde SUBST é a substituição usual de uma lista vinculativa e SUB(x, y, m) significa substituir x por y em todos os lugares em que x ocorre dentro de m.

A regra também pode ser estendida para tratar cláusulas não unitárias nas quais aparece um literal de igualdade:

- **Paramodulação:** Para quaisquer termos x, y e z, onde z aparece em algum lugar no literal m_i e onde UNIFICAR(x, z) = θ,

$$\frac{\ell_1 \vee \cdots \vee \ell_k \vee x = y, \qquad\qquad m_1 \vee \cdots \vee m_n}{\text{SUB}(\text{SUBST}(\theta, x), \text{SUBST}(\theta, y), \text{SUBST}(\theta, \ell_1 \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_n)} .$$

Por exemplo, de

$$P(F(x, B), x) \vee Q(\mathrm{x}) \qquad \text{e} \qquad F(A,y)=y \vee R(y)$$

temos θ = UNIFICAR($F(A, y)$, $F(x, B)$)= $\{x/A, y/B\}$, e podemos concluir através de paramodulação a sentença:

$$P(B, A) \vee Q(A) \vee R(B).$$

A paramodulação gera um procedimento de inferência completo para lógica de primeira ordem com igualdade.

Uma terceira abordagem trata o raciocínio de igualdade inteiramente em um algoritmo de unificação estendido. Isto é, os termos são unificáveis se são *comprovadamente* iguais sob alguma substituição, onde a palavra "comprovadamente" permite raciocínio com igualdade. Por exemplo, os termos 1 + 2 e 2 + 1 normalmente não são unificáveis, mas um algoritmo de unificação que saiba que $x + y = y + x$ poderia unificá-los com a substituição vazia. A **unificação equacional** desse tipo pode ser feita com algoritmos eficientes, criados para os axiomas específicos utilizados (comutatividade, associatividade, e assim por diante), em vez de utilizar a inferência explícita com esses axiomas. Os provadores de teoremas que empregam essa técnica estão intimamente relacionados aos sistemas PLR descritos na Seção 9.4.

9.5.6 Estratégias de resolução

Sabemos que aplicações repetidas da regra de inferência de resolução eventualmente encontrarão uma prova, se existir alguma. Nesta subseção, examinaremos estratégias que ajudam a encontrar

provas de maneira *eficiente*.

Preferência unitária: Essa estratégia tem preferência por resoluções em que uma das sentenças é um único literal (também conhecida como **cláusula unitária**). A ideia por trás da estratégia é a de que estamos tentando produzir uma cláusula vazia e, assim, poderia ser boa ideia preferir inferências que produzissem cláusulas mais curtas. A resolução de uma sentença unitária (como P) com qualquer outra sentença (como $\neg P \vee \neg Q \vee R$) sempre gera uma cláusula (nesse caso, $\neg Q \vee R$) mais curta que a outra cláusula. Quando foi experimentada pela primeira vez na inferência proposicional em 1964, a estratégia de preferência unitária levou a uma aceleração drástica, tornando possível provar teoremas que não podiam ser manipulados sem a preferência. A **resolução unitária** é uma forma restrita de resolução, na qual toda etapa de resolução deve envolver uma cláusula unitária. A resolução unitária é incompleta no caso geral, mas é completa para bases de conhecimento de Horn. As provas de resolução unitária em bases de conhecimento de Horn se assemelham ao encadeamento para a frente.

O provador de teorema OTTER (Organhized Techniques for Theorem-proving and Effective Research — técnicas organizadas para prova de teorema e busca eficaz, McCune, 1992), utiliza uma forma de busca pela melhor escolha. Sua função heurística mede o "peso" de cada cláusula, onde as cláusulas mais leves são as preferidas. A escolha exata da heurística depende do usuário, mas, geralmente, o peso de uma cláusula deve estar correlacionado com o seu tamanho ou dificuldade. As cláusulas unitárias são tratadas como leves; a busca pode ser vista então como uma generalização da estratégia de preferência pela unidade.

Conjunto de apoio: As preferências que experimentam primeiro certas resoluções são úteis mas, em geral, é mais eficiente tentar eliminar por completo algumas resoluções potenciais. Por exemplo, podemos insistir que cada etapa de resolução envolva pelo menos um elemento de um conjunto especial de cláusulas — o *conjunto de apoio*. O resolvente é então adicionado ao conjunto de apoio. Se o conjunto de apoio for pequeno em relação à base de conhecimento inteira, o espaço de busca será drasticamente reduzido.

Temos de ser cuidadosos com essa abordagem porque uma escolha ruim para o conjunto de apoio tornará o algoritmo incompleto. Porém, se escolhermos o conjunto de apoio S de forma que as sentenças restantes sejam satisfatíveis em conjunto, a resolução pelo conjunto de apoio será completa. Por exemplo, pode-se usar a consulta negada como conjunto de apoio, no pressuposto de que a base de conhecimento original seja consistente (afinal, se ela não for consistente, o fato de que a consulta segue dela será vazio). A estratégia de conjunto de apoio tem a vantagem adicional de gerar árvores de prova orientadas para objetivos que frequentemente são fáceis de ser compreendidas pelos seres humanos.

Resolução de entrada: Nessa estratégia, toda resolução combina uma das sentenças de entrada (a partir da BC ou da consulta) com alguma outra sentença. A prova da Figura 9.11 emprega apenas resoluções de entrada e tem a forma característica de uma única "espinha dorsal" com sentenças isoladas combinando-se com a espinha. É claro que o espaço de árvores de prova com essa forma é menor que o espaço de todos os grafos de prova. Em bases de conhecimento de Horn, o *Modus Ponens* é uma espécie de estratégia de resolução de entrada porque combina uma implicação da BC

original com algumas outras sentenças. Desse modo, não surpreende que a resolução de entrada seja completa para bases de conhecimento que estão em forma de Horn, mas seja incompleta no caso geral. A estratégia de **resolução linear** é uma ligeira generalização que permite a resolução conjunta de P e Q se P está na BC original ou se P é um ancestral de Q na árvore de prova. A resolução linear é completa.

Subordinação: O método de **subordinação** elimina todas as sentenças que são subordinadas por uma sentença existente na BC (isto é, são mais específicas que ela). Por exemplo, se $P(x)$ está na BC, não há sentido em adicionar $P(A)$ e faz até mesmo menos sentido adicionar $P(A) \lor Q(B)$. A subordinação ajuda a manter a BC pequena, e isso ajuda a manter pequeno o espaço de busca.

Usos práticos de resolução de provadores de teoremas

Os provadores de teoremas podem ser aplicados aos problemas envolvidos na **verificação** e na **síntese** de hardware e software. Desse modo, a pesquisa em provas de teoremas se desenvolveu nos campos de projeto de hardware, linguagens de programação e engenharia de software — não apenas em IA.

No caso do hardware, os axiomas descrevem as interações entre sinais e elementos de circuitos (veja a Seção 8.4.2 para um exemplo). Raciocinadores lógicos projetados especialmente para verificação foram capazes de verificar CPUs inteiras, incluindo suas propriedades de sincronização (Srivas e Bickford, 1990). O provador de teoremas AURA foi aplicado para projetar circuitos que são mais compactos do que qualquer projeto anterior (Wojciechowski e Wojcik, 1983).

No caso do software, o raciocínio sobre programas é bastante semelhante ao raciocínio sobre ações, como no Capítulo 7: axiomas descrevem as precondições e os efeitos de cada declaração. A síntese formal de algoritmos foi um dos primeiros usos de provadores de teoremas, como descrito por Cordell Green (1969a), que se baseou nas ideias anteriores de Herbert Simon (1963). A ideia é provar um teorema construtivamente no sentido de que "existe um programa p satisfazendo determinada especificação". Embora a **síntese dedutiva** totalmente automatizada, como é chamada, não tenha ainda se tornado viável para programação de propósito geral, as sínteses dedutivas guiadas à mão têm sido bem-sucedidas na concepção de vários algoritmos novos sofisticados. A síntese de programas com propósitos especiais também é uma área prática de pesquisa, como o código de computação científica.

Já estão sendo aplicadas técnicas similares para verificação de software por sistemas como o verificador de modelos SPIN (Holzmann, 1997). Por exemplo, o programa de controle de naves espaciais Remote Agent foi verificado antes e após o voo (Havelund *et al*., 2000). O algoritmo de criptografia de chave pública RSA e o algoritmo de correspondência de sequências de Boyer-Moore foram verificados dessa maneira (Boyer e Moore, 1984).

9.6 RESUMO

Apresentamos uma análise da inferência lógica em lógica de primeira ordem e uma série de algoritmos para realizá-la.

- Uma primeira abordagem utiliza regras de inferência (**instanciação universal** e **instanciação existencial**) para **proposicionalizar** o problema de inferência. Em geral, essa abordagem é muito lenta, a menos que o domínio seja pequeno.
- O uso da **unificação para i**dentificar substituições apropriadas para variáveis elimina a etapa de instanciação em provas de primeira ordem, tornando o processo muito mais eficiente em muitos casos.
- Uma versão elevada de ***Modus Ponens*** usa a unificação para fornecer uma regra de inferência natural e poderosa, o ***Modus Ponens* generalizado**. Os algoritmos de **encadeamento para a frente** e **encadeamento para trás** aplicam essa regra a conjuntos de cláusulas definidas.
- O *Modus Ponens* generalizado é completo para cláusulas definidas, embora o problema de consequência lógi*ca seja* **semidecidível**. No caso de bases de conhecimento **Datalog** que consistem em cláusulas definidas livres de funções, a consequência lógica é decidível.
- O encadeamento para a frente é usado em **bancos de dados dedutivos**, onde pode ser combinado a operações de bancos de dados relacionais. Ele também é utilizado em **sistemas de produção** que executam atualizações eficientes com conjuntos de regras muito grandes. O encadeamento para a frente é completo para programas Datalog e funciona em tempo polinomial.
- O encadeamento para trás é usado em **sistemas de programação em lógica** que empregam tecnologia sofisticada de compiladores para produzir uma inferência muito rápida. O encadeamento para trás se ressente de inferências redundantes e laços infinitos; esses problemas podem ser atenuados por **memoização**.
- O Prolog, ao contrário da lógica de primeira ordem, usa um mundo fechado com a suposição de nomes únicos e negação por falha. Isso faz do Prolog uma linguagem de programação mais prática, mas a leva longe da lógica pura.
- A regra de inferência de **resolução** generalizada produz um sistema de prova completo para a lógica de primeira ordem, utilizando bases de conhecimento em forma normal conjuntiva.
- Existem várias estratégias para reduzir o espaço de busca de um sistema de resolução sem comprometer a completude. Uma das questões mais importantes é tratar com a igualdade; mostramos como usar a **demodulação** e a **paramodulação**.
- Foram usados provadores de teoremas eficientes baseados em resolução para provar teoremas matemáticos interessantes e para verificar e sintetizar software e hardware.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Gottlob Frege, que desenvolveu a lógica de primeira ordem completa em 1879, baseou seu sistema de inferência em uma coleção de esquemas logicamente válidos, somada a uma única regra de inferência, o *Modus Ponens*. Whitehead e Russell (1910) expuseram as chamadas *regras de passagem* (a expressão real se deve a Herbrand (1930)), que são usadas para mover quantificadores para o início das fórmulas. As constantes de Skolem e as funções de Skolem foram introduzidas, de forma apropriada, por Thoralf Skolem (1920). Curiosamente, foi Skolem quem introduziu o universo Herbrand (Skolem, 1928).

O teorema de Herbrand (Herbrand, 1930), desempenhou uma função vital no desenvolvimento de métodos automatizados de raciocínio. Herbrand também é considerado o inventor da unificação.

Gödel (1930) se baseou nas ideias de Skolem e Herbrand para mostrar que a lógica de primeira ordem tem um procedimento de prova completo. Alan Turing (1936) e Alonzo Church (1936) mostraram simultaneamente, utilizando provas muito diferentes, que a validade em lógica de primeira ordem não era decidível. O excelente texto de Enderton (1972) explica todos esses resultados de modo rigoroso, porém compreensível.

Foi Abraham Robinson quem propôs que um raciocinador automático poderia ser construído usando proposicionalização e o teorema de Herbrand, e foi Paul Gilmore (1960) quem escreveu o primeiro programa. Davis e Putnam (1960) introduziram o método de proposicionalização da Seção 9.1. Prawitz (1960) desenvolveu a ideia-chave de permitir que a busca de inconsistência proposicional orientasse o processo de busca e de gerar termos do universo de Herbrand apenas quando fosse necessário estabelecer a inconsistência proposicional. Depois de um desenvolvimento adicional por outros pesquisadores, essa ideia levou J. A. Robinson (sem relação com o outro pesquisador) a desenvolver o método de resolução (Robinson, 1965).

Em IA, a resolução foi adotada em sistemas de perguntas e respostas por Cordell Green e Bertram Raphael (1968). As primeiras implementações de IA dedicavam um grande esforço a estruturas de dados que permitiriam a recuperação eficiente de fatos; esse trabalho foi abordado em textos de programação em IA (Charniak *et al*., 1987; Norvig, 1992; Forbus e de Kleer, 1993). No início da década de 1970, o **encadeamento para a frente** estava bem estabelecido em IA como uma alternativa de fácil compreensão para a resolução. Em geral, as aplicações de IA envolviam grande número de regras; por essa razão, era importante desenvolver uma tecnologia eficiente de correspondência de regras, em particular no caso de atualizações incrementais. A tecnologia de **sistemas de produção** foi desenvolvida para dar suporte a essas aplicações. A linguagem de sistemas de produção OPS-5 (Forgy, 1981; Brownston *et al*., 1985), incorporando o eficiente processo de correspondência ***rete*** (Forgy, 1982), foi usado para aplicações tal como o sistema especialista R1 para configuração de minicomputadores (McDermott, 1982).

A arquitetura cognitiva do SOAR (Laird *et al*., 1987;. Laird, 2008) foi projetada para lidar com conjuntos de regras muito grandes — até um milhão de regras (Doorenbos, 1994). Exemplos de aplicações do SOAR incluem o controle de aviões de combates simulados (Jones *et al*., 1998), gestão do espaço aéreo (Taylor *et al*., 2007), os personagens de IA para jogos de computador (Wintermute *et al*., 2007) e as ferramentas de treinamento para os soldados (Wray e Jones, 2005).

O campo de **bancos de dados dedutivos** começou com um seminário em Toulouse em 1977, que reuniu especialistas em sistemas de inferência lógica e de bancos de dados (Gallaire e Minker, 1978).

O influente trabalho de Chandra e Harel (1980) e de Ullman (1985) levou à adoção de Datalog como linguagem-padrão para bancos de dados dedutivos. O desenvolvimento da técnica de **conjuntos mágicos** para reescrita de regras por Bancilhon *et al*. (1986) permitiu ao encadeamento para a frente tirar proveito da vantagem da orientação a objetivos do encadeamento para trás. O trabalho atual inclui a ideia de integração de múltiplos bancos de dados em um espaço de dados consistente (Halevy, 2007).

O **encadeamento para trás** para inferência lógica surgiu na linguagem PLANNER de Hewitt (1969). Enquanto isso, em 1972, o pesquisador francês Alain Colmerauer havia desenvolvido e

implementado o **Prolog** com a finalidade de analisar a linguagem natural — as cláusulas Prolog foram planejadas inicialmente para serem regras de gramática livres de contexto (Roussel, 1975; Colmerauer *et al*., 1973).

Grande parte da base teórica da programação em lógica foi desenvolvida por Robert Kowalski, trabalhando em conjunto com Colmerauer; consulte Kowalski (1988) e Colmerauer e Roussel (1993) para uma visão geral histórica. Em geral, compiladores Prolog eficientes se baseiam no modelo de computação de máquina abstrata de Warren (WAM — Warren Abstract Machine), desenvolvido por David H. D. Warren (1983). Van Roy (1990) mostrou que os programas Prolog podem ser competitivos em termos de velocidade com programas C.

Os métodos para evitar laços repetitivos desnecessários em programas recursivos de lógica foram desenvolvidos independentemente por Smith *et al*. (1986) e por Tamaki e Sato (1986). Este último artigo também incluía a memoização para programas em lógica, um método desenvolvido extensivamente como **programação em lógica tabulada** por David S. Warren. Swift e Warren (1994) mostram como estender a WAM para manipular a tabulação, permitindo que programas Datalog funcionem com rapidez uma ordem de magnitude maior que os sistemas de encadeamento para a frente de bancos de dados dedutivos.

Os primeiros trabalhos teóricos sobre programação em lógica de restrições foram realizados por Jaffar e Lassez (1987). Jaffar *et al*. (1992a) desenvolveram o sistema CLP(R) para tratar restrições de valores reais. Agora existem produtos comerciais para a resolução de problemas de configuração e otimização em larga escala com programação por restrição; um dos mais conhecidos é o ILOG (Juncker, 2003). A programação por conjunto de respostas (Gelson, 2008) estende o Prolog, permitindo disjunção e negação.

Textos sobre programação em lógica e Prolog incluem Shoham (1994), Bratko (2001), Clocksin (2003) e Clocksin e Mellish (2003). Antes de 2000, o *Journal of Logic Programming* foi o periódico de referência; agora, ele foi substituído por *Theory and Practice of Logic Programming*. As conferências sobre programação em lógica incluem a International Conference on Logic Programming (ICLP) e o International Logic Programming Symposium (ILPS).

A pesquisa em **demonstração de teoremas matemáticos** se iniciou até mesmo antes do desenvolvimento dos primeiros sistemas de primeira ordem completos. O Geometry Theorem Prover de Herbert Gelernter (Gelernter, 1959) usava métodos de busca heurística combinados a diagramas para podar falsos subobjetivos e foi capaz de provar alguns resultados bastante complicados de geometria euclidiana. As regras de demodulação e paramodulação destinadas ao raciocínio com igualdade foram introduzidas por Wos *et al*. (1967) e Wos e Robinson (1968), respectivamente. Essas regras também foram desenvolvidas independentemente no contexto de sistemas de reescrita de expressões (Knuth e Bendix, 1970). A incorporação do raciocínio com igualdade ao algoritmo de unificação se deve a Gordon Plotkin (1972). Jouannaud e Kirchner (1991) apresentam a unificação equacional de uma perspectiva de reescrita de expressões. Baader e Snyder (2001) apresentaram uma visão geral de unificação.

Foram propostas várias estratégias de controle para resolução, começando com a estratégia de preferência unitária (Wos *et al*., 1964). O conjunto de estratégia de suporte foi proposto por Wos *et al*. (1965), a fim de proporcionar certo grau de orientação a objetivos na resolução. A resolução linear surgiu primeiro em Loveland (1970). Genesereth e Nilsson (1987, Capítulo 5) oferecem uma

breve, embora completa, análise de ampla variedade de estratégias de controle.

O livro *A Computational Logic* (Boyer e Moore, 1979) é a referência básica sobre o provador de teoremas de Boyer-Moore. Stickel (1992) focaliza o Prolog Techonogy Theorem Prover (PTTP), que combina as vantagens da compilação de Prolog com a completude da eliminação de modelos. O SETHEO (Letz *et al*., 1992) é outro provador de teoremas extensamente usado, que se baseia nessa abordagem. O LEANTAP (Beckert e Posegga, 1995) é um provador de teoremas eficiente, implementado em apenas 25 linhas de Prolog. Weidenbach (2001) descreve o SPASS, um dos provadores de teoremas mais fortes atualmente. O provador de teorema de maior sucesso nos últimos concursos anuais foi o VAMPIRE (Riazanov e Voronkov, 2002). O sistema COQ (Bertot *et al*., 2004) e o solucionador equacional E (Schulz, 2004) também se mostraram ferramentas valiosas para provar correção. Os provadores de teoremas têm sido usados para sintetizar e verificar o software automaticamente para controle de espaçonaves (Denney *et al*., 2006), incluindo a nova cápsula da Nasa, Orion (Lowry, 2008). O projeto do FM9001, microprocessador de 32 bits, foi provado estar correto pelo sistema NQTHM (Hunt e Brock, 1992). A Conferência sobre a Dedução Automática (CADE) realiza um concurso anual de provadores de teoremas automatizados. De 2002 a 2008, o sistema mais bem-sucedido foi o VAMPIRE (Riazanov e Voronkov, 2002). Wiedijk (2003) compara a força de 15 provadores matemáticos. O TPTP (Milhares de Problemas de Provadores de Teoremas) é uma biblioteca de problemas de provas de teoremas, útil para comparar o desempenho dos sistemas (Sutcliffe e Suttner, 1998; Sutcliffe *et al*., 2006).

Os provadores de teoremas aparecem com resultados matemáticos recentes que escaparam da atenção de matemáticos humanos por décadas, conforme detalhado no livro *Automated Reasoning and the Discovery of Missing Elegant Proofs* (Wos e Pieper, 2003). O programa SAM (matemática semiautomatizada) foi o primeiro, provando um lema na teoria dos reticulados (Guarda *et al*., 1969). O programa AURA também respondeu questões abertas em diversas áreas da matemática (Wos e Winker, 1983). O provador do teorema de Boyer−Moore (Boyer e Moore, 1979) foi utilizado por Natarajan Shankar para dar a primeira prova formal totalmente rigorosa do Teorema da Incompletude de Gödel (Shankar, 1986). O sistema NUPRL provou o paradoxo de Girard (Howe, 1987) e o Lema de Higman (Murthy e Russell, 1990). Em 1933, Herbert Robbins propôs um conjunto de axiomas simples — a **álgebra de Robbins** —, que parecia definir a álgebra booleana, mas nenhuma prova pôde ser encontrada (apesar do trabalho sério de Alfred Tarski e outros). Em 10 de outubro de 1996, após oito dias de computação, o EQP (uma versão do OTTER) encontrou uma prova (McCune, 1997).

Muitos trabalhos em lógica matemática podem ser encontrados em *From Frege to Gödel: A Source Book in Mathematical Logic* (Van Heijenoort, 1967). Os livros didáticos voltados para a dedução automatizada incluem o clássico *Symbolic Logic and Mechanical Theorem Proving* (Chang e Lee, 1973), bem como trabalhos mais recentes por Duffy (1991), Wos *et al.* (1992), Bibel (1993) e Kaufmann *et al.* (2000). A revista principal para prova de teoremas é *Journal of Automated Reasoning;* as conferências principais são as anuais Conference on Automated Reasoning (CADE) e International Joint Conference on Automated Reasoning (IJCAR). O *Handbook of Automated Reasoning* (Robinson e Voronkov, 2001) reúne trabalhos na área. *Mechanizing Proof* (2004) de MacKenzie aborda a história e a tecnologia de prova de teoremas para o público leigo.

EXERCÍCIOS

9.1 Prove que a instanciação universal é correta e que a instanciação existencial produz uma base de conhecimento inferencialmente equivalente.

9.2 A partir de *Gosta*(*Jerry, Sorvete*) parece razoável deduzir ∃*x Gosta*(*x*, *Sorvete*). Enuncie uma regra de inferência geral, a **introdução do existencial**, que sanciona essa inferência. Estabeleça cuidadosamente as condições que devem ser satisfeitas pelas variáveis e os termos envolvidos.

9.3 Suponha que uma base de conhecimento contenha apenas uma sentença, ∃*x TãoAltoQuanto*(*x*, *Everest*). Quais dos resultados a seguir são resultados legítimos da aplicação da instanciação do existencial?

a. *TãoAltoQuanto*(*Everest*, *Everest*).

b. *TãoAltoQuanto*(*Kilimanjaro*, *Everest*).

c. *TãoAltoQuanto*(*Kilimanjaro*, *Everest*) ∧ *TãoAltoQuanto*(*BenNevis*, *Everest*) (depois de duas aplicações).

9.4 Para cada par de sentenças atômicas, forneça o unificador mais geral, se existir:

a. $P(A, B, B)$, $P(x, y, z)$.

b. $Q(y, G(A, B))$, $Q(G(x, x), y)$.

c. *MaisVelho*(*Pai*(*y*), *y*), *MaisVelho*(*Pai*(*x*), *João*).

d. *Conhece*(*Pai*(*y*), *y*), *Conhece*(*x*, *x*).

9.5 Considere os reticulados de subordinação mostrados na Figura 9.2. (Seção 9.2.3)

a. Construa o reticulado correspondente à sentença *Emprega*(*Mãe*(*João*), *Pai*(*Ricardo*)).

b. Construa o reticulado correspondente à sentença *Emprega*(*IBM*, *y*) ("Todo mundo trabalha na IBM"). Lembre-se de incluir toda espécie de consulta que se unifique com a sentença.

c. Suponha que ARMAZENAR faça a indexação de cada sentença sob todo nó em seu reticulado de subordinação. Explique como RECUPERAR deve funcionar quando algumas dessas sentenças contiverem variáveis; utilize como exemplos as sentenças de (a) e (b), e a consulta *Emprega*(*x*, *Pai*(*x*)).

9.6 Anote as representações lógicas para as seguintes sentenças, adequando para uso com *Modus Ponens* generalizado:*

a. Cavalos, vacas e porcos são mamíferos.

b. A prole de um cavalo é um cavalo.

c. Barba Azul é um cavalo.

d. Barba Azul é o pai ou mãe de Charlie.

e. Prole e pai ou mãe são relações inversas.

f. Todo mamífero tem um pai ou mãe.

9.7 Estas questões dizem respeito a questões de substituição e skolemização.

a. Dada a premissa $\forall x\ \exists y\ P(x, y)$, não é válido concluir que $\exists qP(\theta, \theta)$. Dê um exemplo de predicado P em que o primeiro é verdadeiro, mas o segundo é falso.

b. Suponha que um mecanismo de inferência seja escrito incorretamente com a verificação de ocorrência omitida, de modo que permite a um literal como $P(x, F(x))$ ser unificado com $P(\theta, \theta)$ (como mencionado, a maioria das implementações de Prolog realmente permite isso). Mostre que tal mecanismo de inferência vai permitir a conclusão $\exists y\ P(\theta, \theta)$ a ser inferida a partir da premissa $\forall x\ \exists y\ P(x, y)$.

c. Suponha que um procedimento que converte lógica de primeira ordem para forma clausal incorretamente skolemiza $\forall x\ \exists y\ P(x, y)$ para P(x, Sk0), isto é, substitui y por uma constante Skolem em vez de uma função Skolem de x. Mostre que um mecanismo de inferência que usa tal procedimento vai permitir igualmente que $\exists qP(\theta, \theta)$ seja inferido da premissa $\forall x\ \exists y\ P(x, y)$.

d. Um erro comum entre os estudantes é supor que, na unificação, seja permitido substituir um termo por uma constante Skolem em vez de uma variável. Por exemplo, eles vão dizer que as fórmulas $P(Sk1)$ e $P(A)$ podem ser unificadas sob a substituição {Sk1/A}. Dê um exemplo onde isso leva a uma inferência inválida.

9.8 Explique como escrever qualquer problema 3-SAT de tamanho arbitrário usando uma única cláusula definida de primeira ordem e não mais de 30 fatos básicos (sem variáveis).

9.9 Suponha que sejam dados os seguintes axiomas:

1. $0 \leq 3.$
2. $7 \leq 9.$
3. $\forall x \quad x \leq x.$
4. $\forall x \quad x \leq x + 0.$
5. $\forall x \quad x + 0 \leq x.$
6. $\forall x, y \quad x + y \leq y + x.$
7. $\forall w, x, y, z \quad w \leq y \land x \leq z \ \Rightarrow\ w + x \leq y + z.$
8. $\forall x, y, z \quad x \leq y \land y \leq z \ \Rightarrow\ x \leq z$

a. Dê uma prova por encadeamento para trás da sentença $7 \leq 3 + 9$. (Certifique-se, é claro, de usar apenas os axiomas dados aqui, como se não conhecesse mais nada sobre aritmética.) Mostre apenas as etapas que levam ao sucesso, e não as etapas irrelevantes.

b. Dê uma prova por encadeamento para frente da sentença $7 \leq 3 + 9$. Mais uma vez, mostre apenas as etapas que levam ao sucesso.

9.10 Uma charada popular entre as crianças é: “Irmãos e irmãs não tenho nenhum, mas o pai desse homem é o filho de meu pai.” Utilize as regras do domínio de família (veja a Seção 8.3.2) para mostrar quem é esse homem. Você pode aplicar qualquer dos métodos de inferência descritos neste capítulo. Por que considera esse enigma difícil?

9.11 Suponha que inserimos em uma base de conhecimento lógico um segmento de dados do censo dos Estados Unidos listando a idade, a cidade de residência, a data de nascimento e a mãe de toda pessoa utilizando os números do seguro social como constantes de identificação para cada uma. Desse modo, a idade de George será dada por *Idade*(443-65-1282, 56). Quais dos esquemas de

indexação S1−S5 seguintes permitirão uma solução eficiente para cada uma das consultas Q1−Q4 (supondo-se o encadeamento para trás normal)?

- **S1:** Um índice para cada átomo em cada posição.
- **S2:** Um índice para cada primeiro argumento.
- **S3:** Um índice para cada átomo de predicado.
- **S4:** Um índice para cada *combinação* de predicado e primeiro argumento.
- **S5:** Um índice para cada *combinação* de predicado e segundo argumento, e um índice para cada primeiro argumento.
- **Q1:** *Idade*(443-44-4321, *x*)
- **Q2:** *ResideEm*(*x*, *Houston*)
- **Q3:** *Mãe*(*x*, *y*)
- **Q4:** *Idade*(*x*, 34) ∧ *ResideEm*(*x*, *PequenaCidadeDosEstadosUnidos*)

9.12 Seria possível supor que podemos evitar o problema de conflito de variáveis na unificação durante o encadeamento para trás padronizando separadamente todas as sentenças na base de conhecimento de uma vez por todas. Mostre que, para algumas sentenças, essa abordagem não pode funcionar. (*Sugestão*: Considere uma sentença da qual uma parte se unifica com outra.)

9.13 Nesta questão, utilizaremos as sentenças que você escreveu no Exercício 9.6 para responder a uma pergunta que utiliza um algoritmo de encadeamento para trás.

a. Desenhe a árvore de prova gerada por um algoritmo de encadeamento para trás exaustivo para a consulta ∃ *h Cavalo*(*h*), onde as cláusulas são comparadas na ordem dada.

b. O que você nota a respeito desse domínio?

c. Quantas soluções para *h* realmente seguem de suas sentenças?

d. Você conseguiria imaginar um meio de encontrar todas elas? (*Sugestão*: consulte Smith *et al*. (1986).)

9.14 Acompanhe a execução do algoritmo de encadeamento para trás da Figura 9.6 (Seção 9.4.1), quando ele é aplicado para resolver o problema de crime (Seção 9.3.1). Mostre a sequência de valores usada pela variável *objetivos* e organize esses valores em uma árvore.

9.15 O código Prolog a seguir define um predicado P (lembre-se de que termos com maiúsculas são variáveis, não constantes em Prolog).

```
P(X, [X|Y]).
P(X, [Y|Z]) :- P(X, Z).
```

a. Mostre as árvores de prova e as soluções correspondentes às consultas `P(A,[2,1,3])` e `P(2,[1,A,3])`.

b. Que operação de lista padrão `P` representa?

9.16 Este exercício, refere-se à ordenação em Prolog.

a. Escreva cláusulas de Prolog que definam o predicado `ordenada(L)`, verdadeiro se e somente se a lista `L` está ordenada em ordem ascendente.

b. Escreva uma definição em Prolog para o predicado `perm(L,M)`, verdadeiro se e somente se `L` é uma permutação de `M`.

c. Defina `ordenar(L,M)` (`M` é uma versão ordenada de `L`) utilizando perm e ordenada.

d. Execute `ordenar` sobre listas cada vez mais longas até perder a paciência. Qual é a complexidade de tempo do seu programa?

e. Escreva um algoritmo de ordenação mais rápido, como a ordenação por inserção ou o quicksort em Prolog.

9.17 Este exercício refere-se à aplicação recursiva de regras de reescrita, utilizando a programação em lógica. Uma regra de reescrita (ou **demodulador**, na terminologia do OTTER) é uma equação com uma orientação especificada. Por exemplo, a regra de reescrita $x + 0, \rightarrow x$ sugere a substituição de qualquer expressão que corresponda a $x + 0$ pela expressão x. A aplicação de regras de reescrita é uma parte central dos sistemas equacionais de raciocínio. Usaremos o predicado `reescrever(X,Y)` para representar regras de reescrita. Por exemplo, a regra de reescrita anterior é representada como `reescrever(X+0,X)`. Alguns termos são *primitivos* e não podem mais ser simplificados; desse modo, escreveremos `primitivo(0)` para dizer que 0 é um termo primitivo.

a. Escreva uma definição de um predicado `simplificar(X Y)` que seja verdadeira quando Y for uma versão simplificada de X, ou seja, quando não houver regras de reescrita aplicáveis a qualquer subexpressão de Y.

b. Escreva uma coleção de regras para a simplificação de expressões que envolvem operadores aritméticos e aplique seu algoritmo de simplificação a alguns exemplos de expressões.

c. Escreva uma coleção de regras de reescrita para diferenciação simbólica e utilize essas regras juntamente com suas regras de simplificação para diferenciar e simplificar expressões que envolvam expressões aritméticas, incluindo exponenciação.

9.18 Este exercício considera a implementação de algoritmos de busca em Prolog. Suponha que `sucessor(X, Y)` seja verdadeira quando o estado `Y` é um sucessor do estado `X` e que `objetivo(X)` seja verdadeira quando `X` é um estado objetivo. Escreva uma definição para `resolver(X, P)`, que significa que `P` é um caminho (uma lista de estados) que começa com `X`, termina em um estado objetivo e consiste em uma sequência de etapas válidas definida por `sucessor`. Você descobrirá que a busca em profundidade é a maneira mais fácil de realizar essa tarefa. Qual seria o nível de facilidade de se adicionar o controle de busca heurística?

9.19 Suponha que uma base do conhecimento contenha apenas as seguintes cláusulas Horn de primeira ordem:

$$Ancestral\ (Mãe\ (x),\ x)$$
$$Ancestral\ (x,\ y) \land Ancestral\ (y,\ z) \Rightarrow Ancestral\ (x,\ z)$$

Considere o algoritmo de encadeamento para frente que na j-ésima iteração, termina quando a BC

contém uma sentença que unifica com a consulta, senão adiciona a BC cada sentença atômica que pode ser inferida a partir das sentenças que já estão na BC após a iteração j – 1.

a. Para cada uma das seguintes consultas, informe se o algoritmo irá (1) dar uma resposta (se sim, escreva essa resposta); ou (2) termina sem nenhuma resposta ou (3) nunca termina.

(i) *Ancestral (Mãe(y), João)*

(ii) *Ancestral (Mãe (Mãe(y)), João)*

(iii) *Ancestral (Mãe (Mãe (Mãe(y))), Mãe(y))*

(iv) *Ancestral (Mãe (João), Mãe (Mãe (João)))*

b. Um algoritmo de resolução pode provar a sentença ¬*Ancestral (João, João)* a partir da base de conhecimento inicial? Explique como ou por quê não.

c. Suponha que adicionemos a asserção que $\neg(Mãe(x) = x)$ e aumentemos o algoritmo de resolução com regras de inferência para igualdade. Qual é a resposta agora para (b)?

9.20 Seja $\mathcal{L}$ a linguagem de primeira ordem com um único predicado $S(p, \theta)$, que significa "p barbeia θ". Assuma um domínio de pessoas.

a. Considere a sentença: "Existe uma pessoa *P* que barbeia todos os que não se barbeiam, e somente as pessoas que não se barbeiam." Expresse isso em $\mathcal{L}$.

b. Converta a sentença em (a) para a forma clausal.

c. Construa uma prova por resolução para mostrar que as cláusulas em (b) são inerentemente inconsistentes. (Observação: você não precisa de axiomas adicionais.)

9.21 Como a resolução pode ser usada para mostrar que uma sentença é válida? E não satisfatível?

9.22 Construa um exemplo com duas cláusulas que possam ser resolvidas de duas maneiras diferentes dando dois resultados diferentes.

9.23 A partir de "Cavalos são animais", segue-se que "A cabeça de um cavalo é a cabeça de um animal". Demonstre que essa inferência é válida executando as etapas a seguir:

a. Converta a premissa e a conclusão na linguagem da lógica de primeira ordem. Utilize três predicados: $CabeçaDe(h, x)$ (significando que "h é a cabeça de x"), $Cavalo(x)$ e $Animal(x)$.

b. Negue a conclusão e depois converta a premissa e a conclusão negada para a forma normal conjuntiva.

c. Utilize a resolução para mostrar que a conclusão segue da premissa.

9.24 Aqui estão duas sentenças na linguagem de lógica de primeira ordem:

(A) $\forall x\ \exists y\ (x \geq y)$

(B) $\exists y\ \forall x\ (x \geq y)$

a. Suponha que os valores das variáveis se estendam sobre todo o conjunto dos números naturais 0, 1, 2,…, ∞ e que o predicado "≥" signifique "é maior que ou igual a". Sob essa interpretação, converta (A) e (B) para linguagem natural.

b. (A) é verdadeira sob essa interpretação?

c. (B) é verdadeira sob essa interpretação?

d. B é consequência lógica de A?

e. A é consequência lógica de B?

f. Usando a resolução, tente provar que (A) segue de (B). Faça isso mesmo se achar que A não é consequência lógica de B; continue até a prova falhar e não ser possível prosseguir (se ela falhar). Mostre a substituição de unificação para cada etapa de resolução. Se a prova falhar, explique exatamente onde, como e por que ela falhou.

g. Agora tente provar que (B) segue de (A).

9.25 A resolução pode produzir provas não construtivas para consultas com variáveis e, assim, tivemos de introduzir mecanismos especiais para extrair respostas definidas. Explique por que essa questão não surge no caso de bases de conhecimento que contêm apenas cláusulas definidas.

9.26 Dissemos neste capítulo que a resolução não pode ser usada para gerar todas as consequências lógicas de um conjunto de sentenças. Algum algoritmo pode fazer isso?

---

[1] Não confunda essas substituições com as interpretações estendidas usadas para definir a semântica de quantificadores. A substituição troca uma variável por um termo (um item de sintaxe) para produzir uma nova sentença, enquanto uma interpretação mapeia uma variável para um objeto no domínio.

[2] O *Modus Ponens* generalizado é mais geral do que o *Modus Ponens* no sentido de que os fatos conhecidos e a premissa da implicação precisam corresponder apenas após uma substituição, em vez de exatamente. Por outro lado, o Modus Ponens permite qualquer sentença como premissa, em vez de apenas uma conjunção de sentenças atômicas.

[3] *Rete* é a forma latina de rede.

[4] A palavra **produção** em **sistemas de produção** denota uma regra de condição-ação.

[5] Observe que, se forem fornecidos axiomas de Peano, esses objetivos podem ser resolvidos por inferência dentro de um programa Prolog.

[6] Uma cláusula também pode ser representada como uma implicação com uma conjunção de átomos na premissa e uma disjunção de átomos na conclusão (Exercício 7.13). Isso é chamado de **forma normal implicativa** ou **forma de Kowalski** [especialmente quando é escrita com um símbolo de implicação da direita para a esquerda (Kowalski, 1979)]e, com frequência, é muito mais fácil de ler.

* NR: Novamente optamos por manter o exemplo original do inglês, usando “PaiOuMãe” como tradução de “parent”.

CAPÍTULO 10

Planejamento clássico

Em que vemos como um agente pode tirar proveito da estrutura de um problema para construir planos de ação complexos.

Definimos a IA como o estudo da ação racional, o que significa que o **planejamento** — a elaboração de um plano de ação para atingir determinados objetivos — é uma parte crítica da IA. Vimos até agora dois exemplos de agentes de planejamento: o agente de resolução de problemas baseado em busca do Capítulo 3 e o agente lógico híbrido do Capítulo 7. Neste capítulo apresentaremos uma representação para problemas de planejamento capaz de descrever problemas que não poderiam ser tratados pelas abordagens anteriores.

A Seção 10.1 desenvolve uma linguagem expressiva, ainda que cuidadosamente restrita, para representar problemas de planejamento. A Seção 10.2 mostra como os algoritmos de busca para a frente e para trás podem tirar proveito dessa representação, principalmente por meio de heurísticas precisas que podem ser derivadas automaticamente da estrutura da representação (isso é análogo ao modo como as heurísticas independentes de domínio foram construídas para problemas de satisfação de restrições no Capítulo 6). A Seção 10.3 mostra como uma estrutura de dados chamada de grafo de planejamento pode fazer a busca mais eficiente por um plano. Em seguida, descrevemos algumas das outras abordagens para planejamento e concluímos comparando as diversas abordagens.

Este capítulo aborda ambientes com agentes únicos, totalmente observáveis, determinísticos, estáticos. Os Capítulos 11 e 17 vão abordar ambientes com múltiplos agentes parcialmente observáveis, estocásticos, dinâmicos.

10.1 DEFINIÇÃO DO PLANEJAMENTO CLÁSSICO

O agente de resolução de problemas do Capítulo 3 pode encontrar sequências de ações que resultam em um estado objetivo. Mas lida com representações atômicas de estados e, portanto, necessita de boas heurísticas específicas de domínio para um bom desempenho. O agente lógico proposicional híbrido do Capítulo 7 pode encontrar planos sem heurísticas específicas de domínio porque utiliza heurísticas independentes de domínio baseadas na estrutura lógica do problema. Mas se baseia em inferência proposicional (livre de variável), o que significa que pode ser intratável quando houver muitas ações e estados. Por exemplo, no mundo de wumpus, a simples ação de mover

um passo à frente tem de ser repetida para todas as quatro orientações do agente, por T passos de tempo, e n^2 localizações atuais.

Em resposta a isso, pesquisadores de planejamento determinaram como **representação fatorada** aquela em que o estado do mundo é representado por um conjunto de variáveis. Utilizamos uma linguagem chamada **PDDL**, ou seja, *Planning Domain Description Language*, que nos permite expressar todas as ações $4Tn^2$ com um esquema de ação. Há várias versões de PDDL; selecionamos uma versão simples e alteramos a sua sintaxe para ficar coerente com o resto do livro.[1] Mostraremos agora como PDDL descreve os quatro itens que precisamos para definir um problema de busca: o estado inicial, as ações que estão disponíveis em um estado, o resultado da aplicação de uma ação e o teste de objetivo.

Cada **estado** é representado como uma conjunção de fluentes que são instanciados, ou seja, átomos sem função. Por exemplo, *Pobre* ∧ *Desconhecido* pode representar o estado de um agente infeliz, e um estado em um problema de entrega de pacotes poderia ser $Em(Caminhão_1, Melbourne) \wedge Em(Caminhão_2, Sydney)$. A **semântica de banco de dados** é utilizada: a suposição de mundo fechado significa que quaisquer fluentes não mencionados são falsos, e a suposição de nomes exclusivos significa que $Caminhão_1$ *e* $Caminhão_2$ são distintos. Os fluentes a seguir *não* são permitidos em um estado: $Em(x, y)$ (porque não são instanciados), ¬*Pobre* (porque é uma negação) e *Em*(*Pai*(*Fred*) , *Sydney*) (porque utiliza um símbolo de função). A representação dos estados é cuidadosamente projetada de modo que um estado pode ser tratado como uma conjunção de fluentes, que pode ser manipulada por inferência lógica, ou como um *conjunto* de fluentes, que pode ser manipulado com operações de conjuntos. Algumas vezes, a **semântica de conjunto** é mais fácil de tratar.

As **ações** são descritas por um conjunto de esquemas de ação que implicitamente definem as funções AÇÕES(s) e RESULTADO(s, a) necessárias para fazer uma busca de resolução de problema. Vimos no Capítulo 7 que qualquer sistema para descrição de ação precisa resolver um problema de persistência — dizer o que muda e o que permanece o mesmo como resultado da ação. O planejamento clássico se concentra em problemas nos quais a maioria das ações deixa a maioria das coisas inalteradas. Pense em um mundo que consista em um conjunto de objetos em uma superfície plana. A ação de empurrar um objeto faz com que o objeto altere a sua localização por um vetor Δ. Uma descrição concisa da ação deverá mencionar apenas Δ; não deverá mencionar todos os objetos que permanecem no lugar. PDDL faz isso especificando o resultado de uma ação em termos do que muda; tudo o que permanece o mesmo não é mencionado.

Um conjunto de ações instanciadas (livre de variável) pode ser representado por um **esquema de ação** único. O esquema é uma representação **elevada** — ele eleva o nível de raciocínio da lógica proposicional a um subconjunto restrito de lógica de primeira ordem. Por exemplo, aqui está um esquema de ação para voar em um avião de um local para outro:

Ação (*Voar*(*p*, *de*, *para*),
  PRECOND: *Em*(*p*, *de*) ∧ *Avião*(*p*) ∧ *Aeroporto* (*de*) ∧ *Aeroporto*(*para*)
  EFEITO: ¬*Em*(*p*, *de*) ∧ *Em*(*p*, *para*)).

O esquema consiste no nome da ação, uma lista de todas as variáveis utilizadas no esquema, uma

precondição e um **efeito**. Apesar de não ter dito ainda como o esquema de ação se converte em sentenças lógicas, pense sobre as variáveis como sendo universalmente quantificadas. Somos livres para escolher os valores que quisermos para instanciar as variáveis. Por exemplo, aqui está uma ação instanciada que resulta da substituição de valores para todas as variáveis:

$$\begin{aligned}&Ação(Voar(P_1, SFO, JFK),\\&\quad \text{PRECOND: } Em(P_1, SFO) \wedge Avião(P_1) \wedge Aeroporto\ (SFO) \wedge Aeroporto\ (JFK)\\&\quad \text{EFEITO: } \neg\ Em\ (P_1, SFO) \wedge Em\ (P_1, JFK)).\end{aligned}$$

A precondição e o efeito de uma ação são conjunções de literais (sentenças atômicas positivas ou negadas). A precondição define os estados em que a ação pode ser executada, e o efeito define o resultado da execução da ação. Uma ação *a* pode ser executada no estado *s* se a precondição de *a* for uma consequência lógica de *s*. A consequência lógica também pode ser expressa com a semântica de conjunto: $s \models q$ se e somente se cada literal positivo em θ estiver em *s* e cada literal negado em θ não estiver. Na notação formal dizemos

$$(a \in \text{AÇÕES}(s)) \Leftrightarrow s \models \text{PRECOND}(a),$$

onde qualquer variável em *a* é universalmente quantificada. Por exemplo,

$$\begin{aligned}&\forall p, de, para\ (Voar(p, de, para) \in \text{AÇÕES}(s)) \Leftrightarrow\\&s \models (Em(p, de) \wedge Avião(p) \wedge Aeroporto(de) \wedge Aeroporto(para)).\end{aligned}$$

Dizemos que a ação *a* é **aplicável** no estado *s* se as precondições forem satisfeitas por *s*. Quando um esquema de ação *a* contém variáveis, ele pode ter múltiplas instâncias aplicáveis. Por exemplo, com o estado inicial definido na Figura 10.1, a ação voar pode ser instanciada como $Voar(P_1, SFO, JFK)$ ou como $Voar(P_2, JFK, SFO)$, ambas aplicáveis no estado inicial. Se uma ação *a* tiver *v* variáveis, então, em um domínio com *k* nomes únicos de objetos, no pior caso levará o tempo $O(v^k)$ para encontrar as ações instanciadas aplicáveis.

$$\begin{aligned}&Início(Em(C_1, SFO) \wedge Em(C_2, JFK) \wedge Em(P_1, SFO) \wedge Em(P_2, JFK)\\&\qquad \wedge Carga(C_1) \wedge Carga(C_2) \wedge Avião(P_1) \wedge Avião(P_2)\\&\qquad \wedge Aeroporto(JFK) \wedge Aeroporto(SFO))\\&Objetivo(Em(C_1, JFK) \wedge Em(C_2, SFO))\\&Ação(Carregar(c, p, a),\\&\quad \text{PRECOND: } Em(c, a) \wedge Em(p, a) \wedge Carga(c) \wedge Avião(p) \wedge Aeroporto(a)\\&\quad \text{EFEITO: } \neg\ Em(c, a) \wedge Dentro(c, p))\\&Ação(Descarregar(c, p, a),\\&\quad \text{PRECOND: } Dentro(c, p) \wedge Em(p, a) \wedge Carga(c) \wedge Avião(p) \wedge Aeroporto(a)\\&\quad \text{EFEITO: } Em(c, a) \wedge \neg\ Dentro(c, p))\\&Ação(Voar(p, de, para),\\&\quad \text{PRECOND: } Em(p, de) \wedge Avião(p) \wedge Aeroporto(de) \wedge Aeroporto(para)\\&\quad \text{EFEITO: } \neg\ Em(p, de) \wedge Em(p, para))\end{aligned}$$

Figura 10.1 Descrição PDDL de um problema de planejamento de transporte de carga.

Às vezes queremos **proposicionalizar** um problema PDDL — substituir cada esquema de ação por um conjunto de ações instanciadas e depois utilizar um solucionador proposicional, como SATPLAN para encontrar uma solução. No entanto, isso é impraticável quando *v e k* são grandes.

O **resultado** de executar a ação *a* no estado *s* é definido como um estado *s′* que é representado pelo conjunto de fluentes formado no início com *s*, removendo os fluentes que aparecem como literais negativos nos efeitos da ação (o que chamamos de **lista de exclusão** ou DEL(*a*)), e adicionando os fluentes que são literais positivos nos efeitos da ação (o que chamamos de **lista de adição** ou ADD(*a*)):

$$\text{RESULTADO}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a). \tag{10.1}$$

Por exemplo, com a ação *Voar*(P_1, *SFO*, *JFK*), removeríamos *Em(*P_1*, SFO*) e adicionaríamos *Em(*P_1*, JFK)*. É uma exigência dos esquemas de ação que qualquer variável no efeito também deve aparecer na precondição. Dessa forma, quando a precondição é comparada com o estado *s*, todas as variáveis são unificadas, e o RESULTADO(*s*, *a*) terá, portanto, apenas átomos instanciados. Em outras palavras, os estados instanciados são fechados sob a operação RESULTADO.

Observe também que os fluentes não se referem explicitamente ao tempo, como aconteceu no Capítulo 7. Lá precisávamos de sobrescritos para o tempo e de axiomas de estado sucessor da forma

$$F^{t+1} \Leftrightarrow AçãoCausasF^{t} \vee (F^{t} \wedge \neg AçãoCausaNãoF^{t}).$$

Em PDDL, os tempos e os estados estão implícitos nos esquemas de ação: a precondição sempre se refere ao tempo *t* e o efeito ao tempo *t* + 1.

Um conjunto de esquemas de ação serve como uma definição de *domínio* de planejamento. Um *problema* específico dentro do domínio é definido com a adição de um estado inicial e um objetivo. O **estado inicial** é uma conjunção de átomos instanciados. (Como em todos os estados, utiliza-se a suposição do mundo fechado, o que significa que quaisquer átomos que não são mencionados são falsos.) O **objetivo** é exatamente como uma precondição: uma conjunção de literais (positivos ou negativos) que podem conter variáveis, tais como *Em*(*p*, *SFO*) ∧ *Avião*(*p*). Todas as variáveis são tratadas como existencialmente quantificadas, de modo que esse objetivo é ter qualquer avião em SFO. O problema é resolvido quando podemos encontrar uma sequência de ações que terminam em um estado que satisfaz o objetivo. Por exemplo, o estado *Rico* ∧ *Famoso* ∧ *Miserável* satisfaz o objetivo *Rico* ∧ *Famoso* e o estado *Avião*($Avião_1$) ∧ *Em* ($Avião_1$, *SFO*) satisfaz o objetivo *Em*(*p*, *SFO*) ∧ *Avião* (*p*).

Definimos o planejamento como um problema de busca: temos um estado inicial, uma função AÇÕES, uma função RESULTADO e um teste de objetivo. Veremos alguns exemplos de problemas antes de investigar algoritmos eficientes de busca.

10.1.1 Exemplo: transporte de carga aérea

A Figura 10.1 mostra um problema de transporte de carga aérea envolvendo carregamento e descarregamento de carga e o seu transporte aéreo de um local para outro. O problema pode ser definido com três ações: *Carregar, Descarregar* e *Voar*. As ações afetam dois predicados: *Dentro*(c, p) significa que a carga c está dentro do avião p, e *Em*(x, a) significa que o objeto x (avião ou carga) está no aeroporto a. Observe que alguns cuidados devem ser tomados para garantir que os predicados *Em* são corretamente atualizados. Quando um avião voa de um aeroporto para outro, toda a carga no interior do avião vai com ele. Na lógica de primeira ordem seria fácil quantificar sobre todos os objetos que estão dentro do avião. Mas a PDDL básica não tem um quantificador universal, por isso precisamos de uma solução diferente. A abordagem que utilizamos é dizer que uma unidade de carga deixa de estar *Em* qualquer lugar quando estiver *Dendro de* um avião; a carga só estará *Em* um novo aeroporto quando for descarregada. Assim, *Em* significa realmente "disponíveis para uso em um determinado local". O plano seguinte é uma solução para o problema:

$$[Carregar\ (C_1, P_1, SFO), Voar\ (P_1, SFO, JFK), Descarregar\ (C_1, P_1, JFK),$$
$$Carregar\ (C_2, P_2, JFK), Voar\ (P_2, JFK, SFO), Descarregar\ (C_2, P_2, SFO)].$$

Finalmente, há o problema das ações espúrias, como *Voar*(P_1, *JFK*, *JFK*), que deveria ser um no-op, mas que tem efeitos contraditórios (de acordo com a definição, o efeito incluiria *Em*(P_1, *JFK*) ∧¬ *Em*(P_1, *JFK*)). É comum ignorar tais problemas porque eles raramente permitem que planos incorretos sejam produzidos. A abordagem correta é acrescentar precondições de desigualdade dizendo que os aeroportos *de* e *para* devem ser diferentes; veja um outro exemplo disso na Figura 10.3.

10.1.2 Exemplo: o problema do pneu sobressalente

Considere o problema de trocar um pneu furado (Figura 10.2). O objetivo é ter um bom pneu sobressalente adequadamente colocado no eixo do carro, onde no estado inicial havia um pneu furado no eixo e um bom pneu sobressalente no porta-malas. Para simplificar, a versão do problema é abstrata, sem parafusos de roda grudados ou outras complicações. Há apenas quatro ações: retirar o pneu sobressalente, retirar o pneu furado do eixo, colocar o pneu sobressalente no eixo e deixar o carro abandonado durante a noite. Suponhamos que o carro esteja estacionado em um bairro particularmente ruim, de modo que o resultado de deixá-lo durante a noite é que os pneus vão desaparecer. Uma solução para o problema é [*Remover*(*Furado, Eixo*) , *Remover*(*Sobressalente, Porta-malas*), *Colocar*(*Sobressalente, Eixo*)].

Início(*Pneu*(*Furado*) ∧ *Pneu*(*Sobressalente*) ∧ *Em*(*furado*, *Eixo*) ∧ *Em*(*sobressalente, Porta-malas*))
Objetivo (*Em*(*Sobressalente, Eixo*))
Ação(*Remover*(*obj, loc*),
PRECOND: *Em*(*obj, loc*)
EFEITO: ¬*Em*(*obj, loc*) ∧ *Em*(*obj, Chão*))
Ação(*Colocar*(*t, Eixo*),

PRECOND: *Pneu*(*t*) ∧ *Em*(*t, Chão*) ∧ ¬*Em*(*Furado, Eixo*)
EFEITO: ¬*Em*(*t, Chão*) ∧ *Em*(*t, Eixo*))
Ação(*DeixarDuranteNoite*,
PRECOND:
EFEITO: ¬*Em*(*Sobressalente, Chão*) ∧ ¬*Em*(*Sobressalente, Eixo*) ∧ ¬*Em*(*Sobressalente, Porta-mala*)
∧ ¬*Em*(*Furado, Chão*) ∧ ¬*Em*(*Furado, Eixo*) ∧ ¬*Em*(*Furado, Porta-malas*))

Figura 10.2 O problema simples do pneu sobressalente.

10.1.3 Exemplo: o mundo dos blocos

Um dos domínios do planejamento mais famoso é conhecido como **mundo dos blocos**. Esse domínio consiste em um conjunto de blocos em forma de cubo montados sobre uma mesa.[2] Os blocos podem ser empilhados, mas apenas um bloco pode estar diretamente sobre o outro. Um braço robótico pode pegar um bloco e movê-lo para outra posição, na mesa ou em cima de outro bloco. O braço poderá pegar apenas um bloco de cada vez, por isso não pode pegar um bloco que tenha outro sobre ele. O objetivo será sempre construir uma ou mais pilhas de blocos, especificada em termos de quais blocos estão no topo de quais outros blocos. Por exemplo, um objetivo poderia ser colocar o bloco *A* sobre o *B* e o *B* sobre o *C* (veja a Figura 10.4).

Início(*Sobre*(*A, Mesa*) ∧ *Sobre*(*B, Mesa*) ∧ *Sobre*(*C,A*)
∧ *Bloco*(*A*) ∧ *Bloco* (*B*) ∧ *Bloco* (*C*) ∧ *Livre*(*B*) ∧ *Livre*(*C*))
Objetivo (*Sobre*(*A,B*) ∧ *Sobre*(*B,C*))
Ação(*Mover*(*b, x, y*),
PRECOND: *Sobre*(*b, x*) ∧ *Livre*(*b*) ∧ *Livre*(*y*) ∧ *Bloco*(*b*) ∧ *Bloco* (*y*) ∧ (*b*≠*x*) ∧ (*b*≠*y*) ∧ (*x*≠*y*),
EFEITO: *Sobre*(*b, y*) ∧ *Livre*(*x*) ∧ ¬*Sobre*(*b, x*) ∧ ¬*Livre*(*y*))
Ação(*MoverParaMesa* (*b, x*),
PRECOND: *Sobre*(*b, x*) ∧ *Livre*(*b*) ∧ *Bloco*(*b*) ∧ (*b*≠*x*),
EFEITO: *Sobre*(*b, Mesa*) ∧ *Livre*(*x*) ∧ ¬ *Sobre*(*b, x*))

Figura 10.3 Um problema de planejamento no mundo dos blocos: construção de uma torre de três blocos. Uma solução é a sequência [*MoverParaMesa*(*C, A*), *Mover*(*B, Mesa, C*), *Mover*(*A, Mesa, B*)].

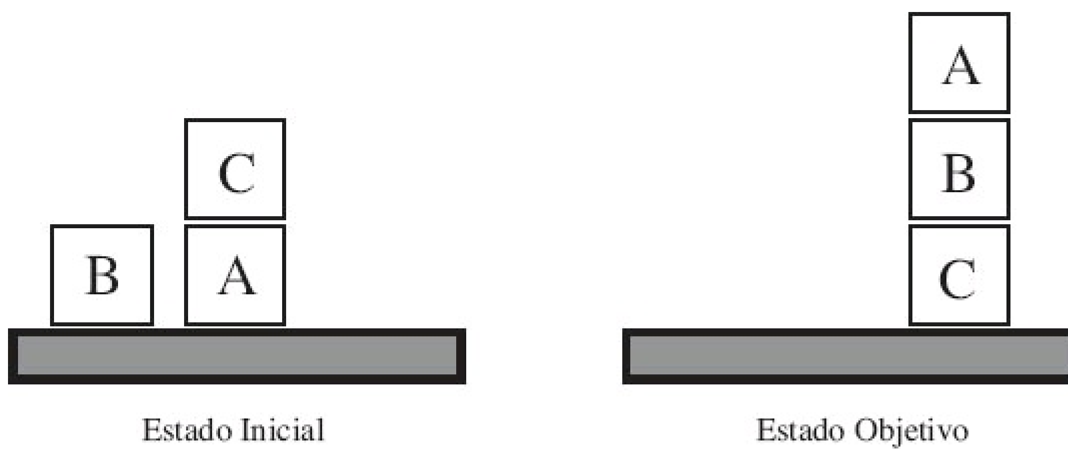


Figura 10.4 Diagrama do problema do mundo dos blocos na Figura 10.3.

Utilizamos *Sobre*(*b, x*) para indicar que o bloco *b* está sobre o bloco *x*, sendo *x* o outro bloco ou a mesa. A ação de mover o bloco *b* do topo de *x* para o topo de *y* será *Mover*(*b, x, y*). Agora, uma das precondições de mover *b* é que nenhum outro bloco está sobre ele. Na lógica de primeira ordem seria $\neg \exists x$ *Sobre*(*x, b*) ou, alternativamente, $\forall x\ \neg$ *Sobre* (*x, b*). PDDL básico não permite quantificadores, então em vez disso introduzimos um predicado *Livre*(*x*) que é verdadeiro quando não há nada sobre *x* (a descrição do problema completo está na Figura 10.3).

A ação *Mover* move o bloco *b* de *x* para *y* se *b* e *y* não tiverem blocos sobre eles. Após a movimentação, *b* ainda está livre, mas *y* não está. A primeira tentativa no esquema *Mover* será

Ação (*Mover*(*b, x, y*),
PRECOND: *Sobre*(*b, x*) ∧ *Livre*(*b*) ∧ *Livre*(*y*),
EFEITO: *Em*(*b, y*) ∧ *Livre*(*x*) ∧ ¬*Sobre*(*b, x*) ∧ ¬*Livre*(*y*)).

Infelizmente, isso não atualiza o fluente *Livre* adequadamente quando *x* ou *y* estiver na mesa. Quando *x* estiver na *Mesa*, essa ação tem o efeito *Livre*(*Mesa*), mas a mesa não deve ser removida; e quando *y* = *Mesa* existe uma precondição *Livre*(*Mesa*)), mas a mesa não deve estar livre para que movamos um bloco sobre ela. Para corrigir isso faremos duas coisas. Primeiro, vamos introduzir uma outra ação para mover um bloco *b* de *x* para a mesa:

Ação (*MoverParaMesa* (*b, x*),
PRECOND: *Sobre* (*b, x*) ∧ *Livre* (*b*),
EFEITO: *Sobre*(*b, Mesa*) ∧ *Livre* (*x*) ∧ ¬ *Sobre* (*b, x*)).

Em segundo lugar, interpretamos *Livre*(*x*) como “há um espaço vago sobre *x* para sustentar um bloco”. Sob essa interpretação, *Livre*(*Mesa*) será sempre verdadeiro. O único problema é que nada impede que o planejador use *Mover*(*b, x, Mesa*) em vez de *MoverParaMesa*(*b, x*). Poderíamos conviver com esse problema — isso conduzirá a um espaço de busca maior do que o necessário, mas não conduzirá a respostas incorretas — ou poderíamos introduzir o predicado *Bloco* e adicionar *Bloco* (*b*) ∧ *Bloco*(*y*) para a precondição de *Mover*.

10.1.4 A complexidade do planejamento clássico

Nesta subseção consideraremos a complexidade teórica do planejamento e a distinção de dois problemas de decisão. **PlanSAT** é a questão de saber se há algum plano que resolva um problema de planejamento. **PlanSAT Limitado** pergunta se existe uma solução de comprimento *k* ou menor; isso pode ser utilizado para encontrar um plano ótimo.

O primeiro resultado é que os dois problemas de decisão são decidíveis para o planejamento clássico. A prova decorre do fato de que o número de estados é finito. Mas, se adicionarmos símbolos de função na linguagem, o número de estados torna-se infinito, e o PlanSAT torna-se apenas semidecidível: existe um algoritmo que resultará na resposta correta para qualquer problema solucionável, mas não poderá terminar em problemas insolúveis. O problema do PlanSAT Limitado permanece decidível mesmo na presença de símbolos de função. Para as provas das afirmações desta seção, consulte Ghallab *et al*. (2004).

Tanto o PlanSAT como o PlanSAT Limitado estão na classe de complexidade PSPACE, uma classe que é maior (e, portanto, mais difícil) do que NP e se refere a problemas que podem ser resolvidos por uma máquina de Turing determinística com uma quantidade de espaço polinomial. Mesmo se fizermos algumas restrições bastante severas, os problemas permanecem bastante difíceis. Por exemplo, se não permitirmos efeitos negativos, os problemas ainda continuam NP-difíceis. No entanto, se também não permitirmos precondições negativas, o PlanSAT reduz para a classe P.

Esses resultados de pior caso podem parecer desanimadores. Podemos nos consolar com o fato de que normalmente não é solicitado que os agentes encontrem planos para quaisquer instâncias de problemas de pior caso, mas apenas para encontrarem planos em domínios específicos (como problemas do mundo dos blocos com *n* blocos), que podem ser muito mais fáceis do que o pior caso teórico. Para muitos domínios (incluindo o mundo dos blocos e o mundo da carga aérea), o PlanSAT Limitado é NP-completo, enquanto o PlanSAT está em P; em outras palavras, o planejamento ótimo é geralmente difícil, mas o planejamento subótimo algumas vezes é fácil. Para se sair bem em problemas mais fáceis do que os piores casos, precisaremos de boas heurísticas de busca. Essa é a verdadeira vantagem do formalismo do planejamento clássico: facilita o desenvolvimento de heurísticas independentes de domínio muito precisas, enquanto os sistemas baseados em axiomas de estado sucessor em lógica de primeira ordem têm tido menos êxito na geração de boas heurísticas.

10.2 ALGORITMOS DE PLANEJAMENTO COMO BUSCA EM ESPAÇO DE ESTADOS

Agora voltaremos nossa atenção para os algoritmos de planejamento. Vimos como a descrição de um problema de planejamento define um problema de busca: podemos pesquisar a partir do estado inicial através do espaço de estados, à procura de um objetivo. Uma das boas vantagens da representação declarativa dos esquemas de ação é que também podemos retroceder a busca do objetivo procurando pelo estado inicial. A Figura 10.5 compara as buscas para a frente e para trás.

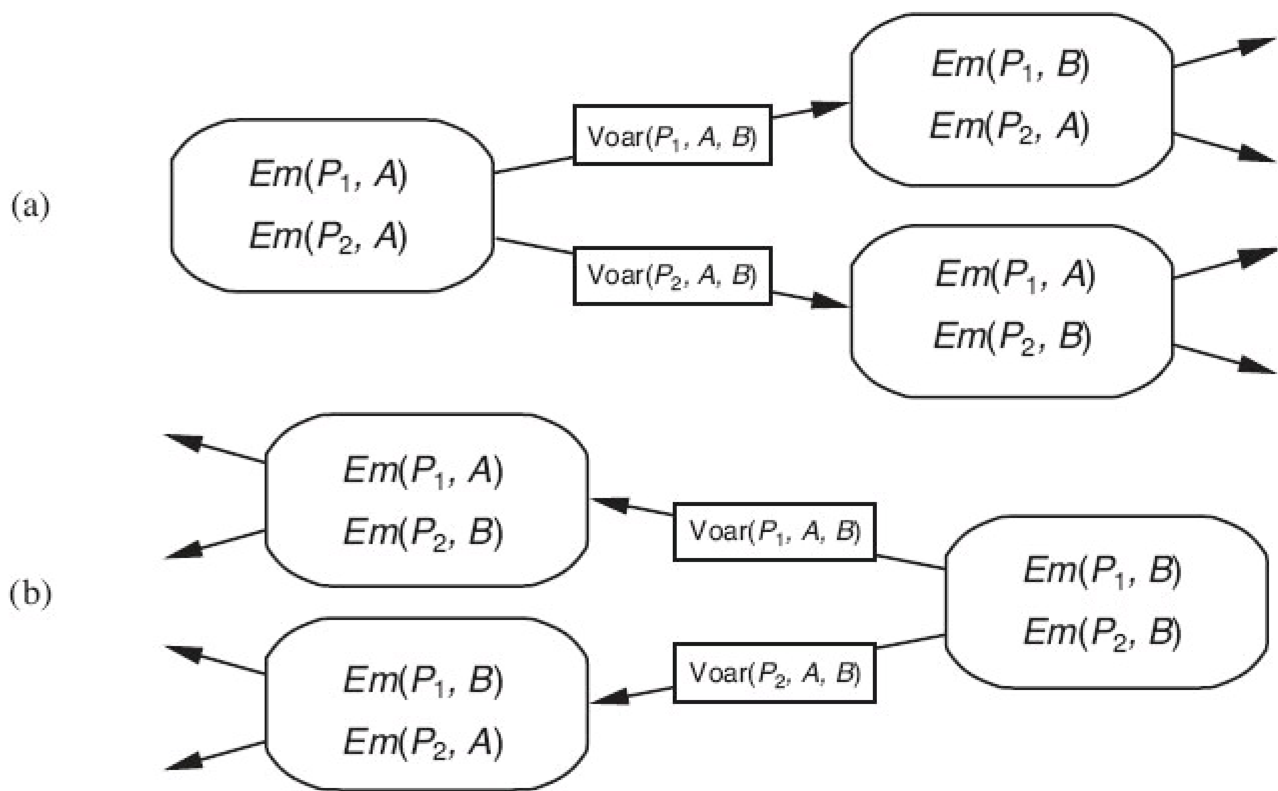


Figura 10.5 Duas abordagens para a busca de um plano. (a) Busca para a frente no espaço de estados (progressão), começando no estado inicial e utilizando as ações do problema para realizar a busca para a frente de um elemento do conjunto de estados objetivos. (b) Busca para trás (regressão) através dos conjuntos de estados relevantes, começando pelo conjunto de estados que representa o objetivo e usando o inverso das ações para procurar para trás pelo estado inicial.

10.2.1 Busca em espaço de estados para a frente (progressão)

Agora que mostramos como um problema de planejamento mapeia em um problema de busca, podemos resolver problemas de planejamento com qualquer um dos algoritmos de busca heurística do Capítulo 3 ou um algoritmo de busca local do Capítulo 4 (desde que acompanhemos as ações utilizadas para atingir o objetivo). Desde os primeiros tempos da pesquisa de planejamento (por volta de 1961) até por volta de 1998, assumiu-se que a busca em espaço de estados para a frente era muito ineficiente para ser prática. Não é difícil explicar as razões.

Em primeiro lugar, a busca para a frente é propensa a explorar ações irrelevantes. Considere a tarefa nobre de comprar uma cópia de *AI: A Modern Approach* de uma livraria on-line. Suponha que exista um esquema de ação *Comprar*(*isbn*), com efeito *Possui*(*isbn*). Os ISBNs têm 10 dígitos; assim, esse esquema de ação representa 10 bilhões de ações instanciadas. Um algoritmo de busca para a frente desinformado teria que começar a enumerar esses 10 bilhões de ações para encontrar uma que leve ao objetivo.

Segundo, os problemas de planejamento muitas vezes têm espaço de estados grandes. Considere um problema de carga aérea em 10 aeroportos, em que cada aeroporto tenha 5 aviões e 20 peças de carga. O objetivo é mover toda a carga do aeroporto *A* para o *B*. Há uma solução simples para o problema: carregar as 20 peças de carga em um dos aviões em *A*, pilotar o avião até *B* e descarregar a carga. A descoberta da solução pode ser difícil porque o fator médio de ramificação é enorme:

cada um dos 50 aviões pode voar para nove outros aeroportos, e cada uma das 200 pacotes pode ser descarregada (se foi carregada) ou carregada em qualquer avião no aeroporto (se foi descarregada). Então, em qualquer estado existe um mínimo de 450 ações (quando todos os pacotes estão nos aeroportos sem aviões) e um máximo de 10.450 (quando todos os pacotes e aviões estão no mesmo aeroporto). Digamos, em média, que haja cerca de 2.000 ações possíveis por estado, então o grafo de busca até a profundidade da solução óbvia tem cerca de 2.000^{41} nós.

Certamente, mesmo essa instância de problema relativamente pequena é impossível sem uma heurística precisa. Embora muitas aplicações de planejamento do mundo real tenham contado com heurísticas específicas de domínio, verifica-se (como veremos na Seção 10.2.3) que se pode derivar automaticamente heurísticas fortes independentes de domínio; isso é o que torna viável a busca para a frente.

10.2.2 Busca para trás (regressão) de estados relevantes

Na busca para trás começamos no objetivo e retrocedemos com as ações até que encontramos uma sequência de passos que alcançam o estado inicial. Chama-se busca de **estados relevantes** porque consideramos apenas as ações que são relevantes ao objetivo (ou estado atual). Como na busca de estado de crença (Seção 4.4), existe um *conjunto* de estados relevantes a considerar em cada passo, não apenas um único estado.

Começamos com o objetivo, que é uma conjunção de literais formando uma descrição de um conjunto de estados, por exemplo, o objetivo $\neg Pobre \wedge Famoso$ descreve os estados em que *Pobre* é falso, *Famoso* é verdadeiro e qualquer outro fluente pode ter qualquer valor. Se houver n fluentes instanciados em um domínio, existem 2^n estados instanciados (cada fluente pode ser verdadeiro ou falso), mas 3^n descrições de conjuntos de estados objetivos (cada fluente pode ser positivo, negativo ou não mencionado).

Em geral, a busca para trás só funciona quando sabemos como regredir a partir de uma descrição de estado para a descrição do estado predecessor. Por exemplo, é difícil buscar para trás uma solução para o problema das n-rainhas porque não há maneira fácil de descrever os estados estão um movimento distante do objetivo. Felizmente, a representação PDDL foi projetada para facilitar a regressão de ações — se um domínio puder ser expresso em PDDL, podemos fazer uma busca nele em regressão. Dada a descrição de objetivo instanciado g e uma ação instanciada a, a regressão a partir de g através de a fornece uma descrição do estado g' definida por

$$g' = (g - \text{ADD}(a)) \cup Precond(a).$$

Ou seja, os efeitos que foram adicionados pela ação não precisam ter sido verdadeiros anteriormente, e também as precondições devem valer antes ou, então, a ação não poderia ter sido executada. Observe que DEL(a) não aparece na fórmula porque, enquanto sabemos que os fluentes em DEL(a), não são mais verdadeiros após a ação, não sabemos se eram ou não verdadeiros antes, então não há nada a ser dito sobre eles.

Para obter o máximo de proveito da busca para trás, temos que lidar com ações e estados

parcialmente instanciados, não totalmente instanciados. Por exemplo, suponha que o objetivo seja entregar uma peça de carga específica em SFO: $Em(C_2, SFO)$. Isso sugere a ação $Descarregar(C_2, p', SFO)$:

$$\begin{aligned}&Ação\ (Descarregar(C_2, p', SFO),\\ &\quad \text{PRECOND: } Em(C_2, p') \wedge Em(p', SFO) \wedge Carga(C_2) \wedge Avião(p') \wedge Aeroporto(SFO)\\ &\quad \text{EFEITO: } Em(C_2, SFO) \wedge \neg Em(C_2, p').\end{aligned}$$

[Observe que **padronizamos** os nomes das variáveis (mudando *p* para *p′* nesse caso) para que não haja confusão entre os nomes de variáveis, se usarmos o mesmo esquema de ação duas vezes em um plano. A mesma abordagem foi usada no Capítulo 9 para inferência lógica de primeira ordem.] Isso representa descarregar o pacote de um avião *não especificado* em SFO; pode ser em qualquer avião, não precisamos dizer agora de qual. Podemos tirar proveito do poder das representações de primeira ordem: uma única descrição resume a possibilidade de usar *qualquer* um dos aviões quantificando implicitamente sobre *p′*. A descrição do estado de regressão é

$$g' = Em\ (C_2, p') \wedge Em\ (p', SFO) \wedge Carga\ (C_2) \wedge Avião\ (p') \wedge Aeroporto\ (SFO).$$

A última questão é decidir quais ações são candidatas a regredir. Na direção para a frente escolhemos ações que são **aplicáveis** — aquelas ações que poderiam ser o próximo passo no plano. Na busca para trás queremos ações que sejam **relevantes** — aquelas ações que poderiam ser o *último* passo em um plano que conduz ao estado objetivo atual.

Para uma ação ser relevante para um objetivo, obviamente deve contribuir para o objetivo: pelo menos um dos efeitos da ação (positivo ou negativo) deve unificar com um elemento do objetivo. O que é menos óbvio é que a ação não deve ter quaisquer efeitos (positivos ou negativos) que negue um elemento do objetivo. Agora, se o objetivo for $A \wedge B \wedge C$ e uma ação tiver o efeito $A \wedge B \wedge \neg C$, então há um sentido coloquial em que essa ação é muito relevante para o objetivo — isso nos leva a dois terços do caminho. Mas ela não é relevante no sentido técnico definido aqui porque essa ação não poderia ser o *passo* final de uma solução — precisaríamos sempre de pelo menos mais um passo para alcançar *C*.

Dado o objetivo $Em(C_2, SFO)$, diversas instanciações de *Descarregar* são relevantes: poderíamos escolher qualquer avião específico para descarregar ou poderíamos deixar o avião não especificado utilizando a ação $Descarregar(C_2, p', SFO)$. Podemos reduzir o fator de ramificação sem excluir quaisquer soluções, sempre utilizando a ação formada pela substituição do unificador mais geral no esquema de ação (padronizado).

Como outro exemplo, considere o objetivo *Possui*(0136042597), dado um estado inicial com 10 bilhões de ISBNs e o esquema de ação único

$$A = Ação(Comprar(i), \text{PRECOND:}ISBN(i), \text{EFEITO:}Possui(i)).$$

Como mencionamos anteriormente, a busca para a frente sem uma heurística teria de começar enumerando 10 bilhões de ações instanciadas *Comprar*. Mas, com a busca para trás, unificaríamos o objetivo *Possui*(0136042597) com o efeito (padronizado) *Possui*(*i'*), produzindo a substituição θ =

{i'/0136042597}. Então poderíamos regredir sobre a ação *Subst*(θ, *A'*) para produzir a descrição do estado predecessor *ISBN*(0136042597). Isso faz parte do estado inicial e, portanto, é uma consequência lógica, e assim concluímos a regressão.

Podemos tornar isso mais formal. Seja a descrição de um objetivo g que contenha um literal objetivo g_i e um esquema de ação A, padronizado para produzir A'. Se A' tiver um efeito literal e'_j onde $Unificar(g_i, e'_j) = \theta$ e onde definimos $a' = \text{SUBST}(\theta, A')$, e se não houver efeito em a' que seja a negação de uma literal em g, então a' será uma ação relevante em direção a g.

A busca para trás mantém o fator de ramificação mais baixo do que a busca para a frente, para a maioria dos domínios de problemas. No entanto, o fato de que a busca para trás utiliza conjuntos de estado, em vez de estados individuais, torna mais difícil chegar a uma boa heurística. Essa é a principal razão pela qual a maioria dos sistemas atuais preferem a busca para a frente.

10.2.3 Heurísticas de planejamento

Nem a busca para a frente ou para trás é eficiente sem uma boa função heurística. Lembre-se do Capítulo 3, em que uma função heurística $h(s)$ estima a distância de um estado s para o objetivo e onde se pode derivar uma heurística **admissível** para essa distância — uma que não superestime —. então podemos utilizar uma busca A* para encontrar as melhores soluções. Uma heurística admissível pode ser obtida através da definição de um **problema relaxado** que é mais fácil de resolver. O custo exato de uma solução para esse problema mais fácil torna-se então a heurística do problema original.

Por definição, não há como analisar um estado atômico e, portanto, requer alguma engenhosidade para um analista humano definir uma boa heurística de domínio específico para problemas de busca com estados atômicos. Planejamento utiliza uma representação fatorada para os estados e esquemas de ação. Isso torna possível definir uma boa heurística independente do domínio e para os programas aplicarem automaticamente uma heurística independente do domínio para um determinado problema.

Pense em um problema de busca como um grafo onde os nós são estados e as arestas são ações. O problema é encontrar um caminho que ligue o estado inicial ao estado objetivo. Há duas maneiras de relaxar esse problema para torná-lo mais fácil: acrescentando mais arestas ao grafo, tornando-o estritamente mais fácil de encontrar um caminho ou agrupando vários nós juntos, formando uma abstração do espaço de estados que tem menos estados e, assim, é mais fácil de fazer a busca.

Olharemos primeiro a heurística que adiciona arestas ao grafo. Por exemplo, a **heurística ignorar as precondições** remove todas as precondições das ações. Toda ação passa a ser aplicável em cada estado, e qualquer fluente objetivo individual pode ser alcançado em um único passo (se houver uma ação aplicável; se não, o problema é impossível). Isso quase implica que o número de passos necessários para resolver um problema relaxado é o número de objetivos não satisfeitos — quase, mas não exatamente, porque (1) alguma ação pode atingir múltiplos objetivos e (2) algumas ações podem desfazer os efeitos de outras. Para muitos problemas obtém-se uma heurística precisa considerando (1) e ignorando (2). Primeiro, relaxamos as ações removendo todas as precondições e todos os efeitos, exceto aqueles que são literais no objetivo. Então, contamos o número mínimo de

ações necessárias, tais que a união dos efeitos dessas ações satisfaça o objetivo. Esse é um exemplo do problema de **cobertura de conjuntos**. Há um pequeno porém: o problema de cobertura de conjuntos é NP-difícil. Felizmente, um simples algoritmo guloso garante encontrar uma cobertura de conjuntos cujo tamanho está dentro de um fator de log *n* da cobertura mínima verdadeira, onde *n* é o número de literais no objetivo. Infelizmente, o algoritmo guloso perde a garantia de admissibilidade.

Também é possível ignorar apenas precondições s*elecionadas* das ações. Considere o quebra-cabeças das peças deslizantes (quebra-cabeças de oito ou 15 peças) da Seção 3.2. Poderíamos codificar isso como um problema de planejamento que envolve as peças com um único esquema *Deslizar*:

Ação Deslizar$((t, s_1, s_2),$
PRECOND: $Em(t, s_1) \wedge Peça(t) \wedge Vazio(s_2) \wedge Adjacente(s_1, s_2)$
EFEITO: $Em(t, s_2) \wedge Vazio(s_1) \wedge \neg Em(t, s_1) \wedge \neg Vazio(s_2))$

Como vimos na Seção 3.6, se removermos as precondições $Vazio(s_2) \wedge Adjacente(s_1, s_2)$, qualquer peça pode mover-se em uma ação para qualquer espaço e obteremos a heurística do número de peças fora de lugar. Se removermos $Vazio(s_2)$ obteremos a heurística da distância de Manhattan. É fácil verificar como essas heurísticas podem ser derivadas automaticamente a partir da descrição do esquema de ação. A facilidade de manipular os esquemas é a grande vantagem da representação fatorada dos problemas de planejamento, em comparação com a representação atômica dos problemas de busca.

Outra possibilidade é a heurística de **ignorar listas de exclusão**. Assuma por um momento que todos os objetivos e precondições contêm apenas literais[3] positivos. Queremos criar uma versão relaxada do problema original que será mais fácil de resolver e onde o tamanho da solução servirá como uma boa heurística. Podemos fazer isso removendo as listas de exclusão de todas as ações (isto é, removendo todos os literais negativos dos efeitos). Isso faz com que seja possível fazer progresso monotônico em direção ao objetivo — nenhuma ação jamais irá desfazer o progresso feito por outra ação. Acontece que encontrar a solução ótima para esse problema relaxado ainda é NP-difícil, mas pode ser encontrada uma solução aproximada em tempo polinomial por subida de encosta. A Figura 10.6 diagrama parte do espaço de estados para dois problemas de planejamento utilizando a heurística de ignorar listas de exclusão. Os pontos representam os estados, as arestas, as ações, e a altura de cada ponto acima do plano inferior representa o valor heurístico. Os estados no plano inferior são as soluções. Em ambos esses problemas, há um caminho amplo até o objetivo. Não há becos sem saída, então não há necessidade de retroceder; uma busca simples de subida de encosta encontrará facilmente uma solução para esses problemas (embora possa não ser uma solução ótima).

Os problemas relaxados nos deixam com um problema de planejamento simplificado — mas ainda caro — apenas para calcular o valor da função heurística. Muitos problemas de planejamento tem 10^{100} estados ou mais, e relaxar as ações não contribui para reduzir o número de estados. Portanto, daremos atenção agora aos relaxamentos que diminuem o número de estados formando uma **abstração do estado** — um mapeamento muitos-para-um dos estados na representação instanciada do problema para a representação abstrata.

A forma mais fácil de abstração de estado é ignorar alguns fluentes. Por exemplo, considere um

problema de carga aérea em 10 aeroportos, 50 aviões e 200 peças de carga. Cada avião pode estar em um dos 10 aeroportos e cada peça pode estar em um dos aviões ou descarregada em um dos aeroportos. Portanto, há $50^{10} \times 200^{50+10} \approx 10^{155}$ estados. Agora considere um determinado problema nesse domínio em que acontece que todos as peças estão em apenas cinco aeroportos e, todas as peças em um determinado aeroporto têm o mesmo destino. Em seguida, uma abstração útil do problema é a remoção de todos os fluentes *Em* exceto os que envolvem um avião e uma peça em cada um dos cinco aeroportos. Agora existem apenas $5^{10} \times 5^{5+10} \approx 10^{17}$ estados. Uma solução nesse espaço de estado abstrato será mais curta do que uma solução no espaço original (e, portanto, será uma heurística admissível), e a solução abstrata é fácil de estender ao problema original (através da inclusão de ações *Carregar e Descarregar* adicionais).

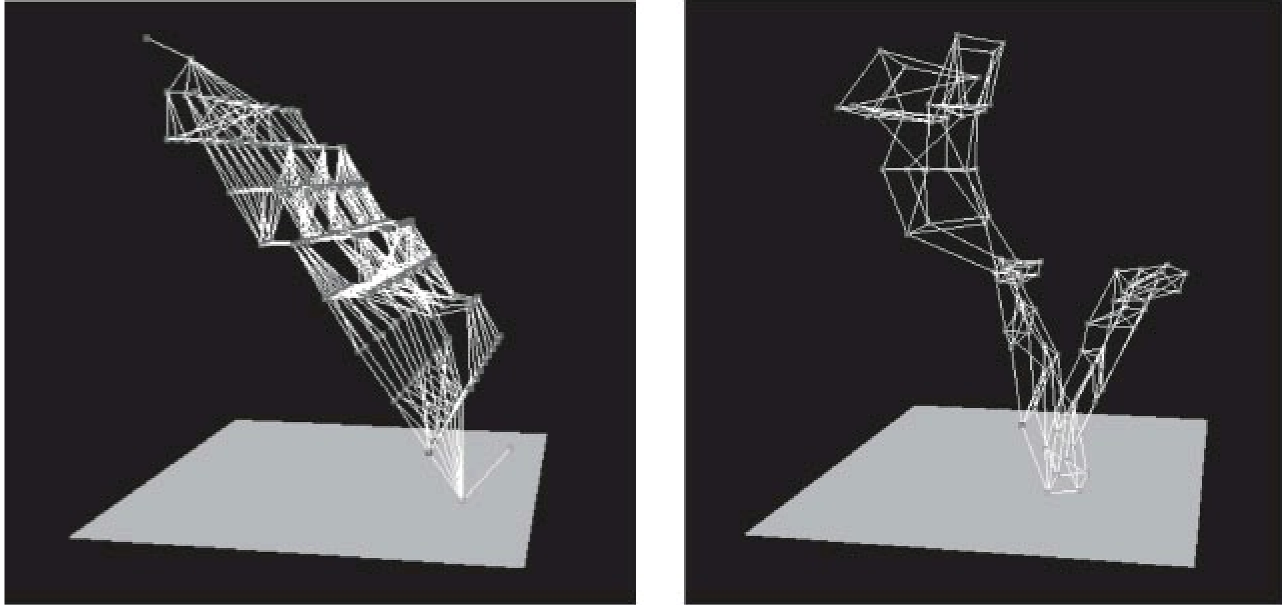


Figura 10.6 Dois espaços de estados de problemas de planejamento com a heurística ignorar listas de exclusão. A altura acima do plano inferior é a pontuação heurística de um estado; estados na parte inferior do plano são objetivos. Não há mínimos locais; assim, a busca do objetivo é simples. De Hoffmann (2005).

A ideia-chave na definição da heurística é a **decomposição**: dividir um problema em partes, resolvendo cada parte de forma independente e depois combinar as partes. A hipótese da **independência de subobjetivos** é que o custo de resolver uma conjunção de subobjetivos é aproximado pela soma dos custos de resolver cada subobjetivo de forma independente. A suposição de independência de subobjetivos pode ser otimista ou pessimista. É otimista quando há interações negativas entre os subplanos para alcançar cada subobjetivo — por exemplo, quando uma ação em um subplano exclui um objetivo alcançado por outro subplano. É pessimista e, portanto, não admissível quando os subplanos contêm ações redundantes — por exemplo, duas ações que poderiam ser substituídas por uma única ação no plano composto.

Suponha que o objetivo seja um conjunto de fluentes G, os quais dividimos em subconjuntos disjuntos $G_1, \ldots, G_n$. Em seguida, encontramos os planos $P_1, \ldots, P_n$, que resolvem os respectivos subobjetivos. Qual é a estimativa de custo do plano para alcançar todos os G? Podemos pensar em cada $Custo(P_i)$ como uma estimativa heurística e sabemos que, se combinarmos as estimativas tomando seus valores máximos, sempre obteremos uma heurística admissível. Então, o max_i

CUSTO(P_i) é admissível, e às vezes é exatamente correto: pode ser que, por acaso, P_1 atinja todos os G_i. Mas, na maioria dos casos, na prática a estimativa é muito baixa. Poderíamos somar os custos em vez disso? Para muitos problemas isso é uma estimativa razoável, mas não é admissível. O melhor caso é quando podemos determinar que G_i e G_j são **independentes**. Se os efeitos de P_i deixarem todas as precondições e objetivos de P_j inalterados, a estimativa CUSTO(P_i) + CUSTO (P_j) é admissível, e mais precisa que a estimativa max. Mostramos, na Seção 10.3.1, que os grafos de planejamento podem ajudar a fornecer melhores estimativas heurísticas.

É claro que existe um grande potencial para reduzir o espaço de busca formando abstrações. O truque é escolher as abstrações corretas e utilizá-las de maneira que faça o custo total — de definir uma abstração, fazer uma busca abstrata e mapear a abstração de volta ao problema original — menor do que o custo de resolver o problema original. A técnica de bancos de dados de padrões da Seção 3.6.3 pode ser útil porque o custo de criação de banco de dados de padrões pode ser amortizado em múltiplas instâncias de problemas.

Um exemplo de um sistema que faz uso de heurísticas eficazes é FF, ou FastForward (Hoffmann, 2005), um buscador em espaço de estados para a frente que usa a heurística de ignorar listas de excluão, estimando a heurística com a ajuda de um grafo de planejamento (veja a Seção 10.3). FF então utiliza a busca de subida de encosta (modificada para construir o plano) com a heurística para encontrar uma solução. Quando atinge um patamar ou máximo local — quando nenhuma ação leva a um estado com melhor pontuação heurística —, o FF utiliza a busca por aprofundamento iterativo até encontrar um estado que seja melhor ou desiste e reinicia a subida de encosta.

10.3 GRAFOS DE PLANEJAMENTO

Todas as heurísticas que sugerimos podem sofrer de imprecisão. Esta seção mostra como uma estrutura de dados especial chamada **grafo de planejamento** pode ser usada para fornecer melhores estimativas de heurísticas. Essas heurísticas podem ser aplicadas a qualquer das técnicas de busca que vimos até agora. Como alternativa, podemos buscar uma solução sobre o espaço formado pelo grafo de planejamento utilizando um algoritmo chamado Graphplan.

A pergunta de um problema de planejamento é se podemos alcançar um estado objetivo do estado inicial. Suponha que seja dada uma árvore de todas as ações possíveis a partir do estado inicial para os estados sucessores, e seus sucessores, e assim por diante. Se indexarmos essa árvore adequadamente, poderemos responder à pergunta de planejamento sobre se "podemos atingir o estado G do estado S_0" imediatamente, olhando para essa árvore. Naturalmente, a árvore é de tamanho exponencial, por isso essa abordagem é impraticável. Um grafo de planejamento é uma aproximação de tamanho polinomial dessa árvore que pode ser construído rapidamente. O grafo de planejamento definitivamente não pode responder se G é acessível de S_0, mas pode *estimar* quantos passos levará para chegar a G. A estimativa é sempre correta quando ela relata que o objetivo não é alcançável e nunca superestima o número de passos, por isso é uma heurística admissível.

Um grafo de planejamento é um grafo direcionado organizado em **níveis**: primeiro, o nível S_0 para

o estado inicial, que consiste de nós que representam cada fluente que é verdadeiro em S_0; depois, o nível A_0, consistindo de nós para cada ação instanciada que poderia ser aplicável em S_0; em seguida, alterna níveis S_i seguidos por A_i até chegar a uma condição de término (a ser discutida mais adiante).

Grosso modo, S_i contém todos os literais que *poderiam* ser verdadeiros no tempo *i*, dependendo das ações executadas nos passos de tempo anteriores. Se for possível *P* ou ¬*P* ser verdade, ambos serão representados em S_i. Também a *grosso modo*, A_i contém todas as ações que *poderiam* ter suas precondições satisfeitas no tempo *i*. Dizemos a *grosso modo* porque o grafo de planejamento registra apenas um subconjunto restrito de interações negativas possíveis entre ações e, portanto, um literal poderia aparecer no nível S_j quando, na verdade, poderia não ser verdadeiro, até num nível posterior, ou nunca aparecer (um literal nunca vai aparecer muito tarde). Apesar do possível erro, erro o nível *j* em que um literal aparece a princípio é uma boa estimativa do quão difícil é atingir o literal a partir do estado inicial.

Os grafos de planejamento funcionam apenas para problemas de planejamento proposicional — aqueles sem variáveis. Como já mencionamos, é simples proposicionalizar um conjunto de esquemas de ações.

Apesar do aumento no tamanho da descrição do problema, os grafos de planejamento têm mostrado ser ferramentas eficazes para resolver problemas de planejamento difícil.

A Figura 10.7 mostra um problema de planejamento simples, e a Figura 10.8 mostra seu grafo de planejamento. Cada ação no nível A_i está conectada a suas precondições em S_i e seus efeitos em S_{i+1}. Assim, um literal aparece porque uma ação o causou, mas também queremos dizer que um literal pode persistir se nenhuma ação negá-lo. Isso é representado pela **ação de persistência** (às vezes chamada *no-op*). Para cada literal *C*, adicionamos uma ação de persistência ao problema com precondição *C* e efeito *C*. O nível A0 na Figura 10.8 mostra uma ação "real", C*omer*(*Bolo*), juntamente com duas ações de persistência desenhadas como pequenas caixas quadradas.

Início(*Ter*(*Bolo*))
Objetivo(*Ter*(*Bolo*) ∧ *Comido*(*Bolo*))
Ação(*Comer*(*Bolo*)
  PRECOND: *Ter*(*Bolo*)
  EFEITO: ¬*Ter*(*Bolo*) ∧ *Comido*(*Bolo*))
Ação(*Assar*(*Bolo*)
  PRECOND: ¬*Ter*(*Bolo*)
  EFEITO: *Ter*(*Bolo*))

Figura 10.7 Problema de "ter bolo e comer bolo".

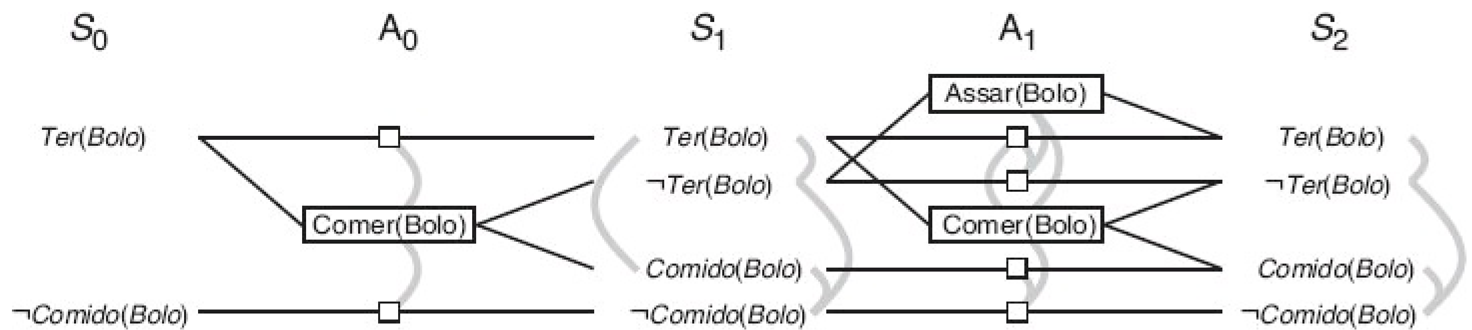


Figura 10.8 O grafo de planejamento para o problema de "ter bolo e comer bolo" até o nível S_2. Os retângulos indicam ações (pequenos quadrados indicam ações de persistência) e linhas retas indicam precondições e efeitos. As ligações de exclusão mútua (ou mutex) estão representados por linhas curvas em cor cinza. Nem todos as ligações de exclusão mútua são mostradas porque o grafo seria muito confuso. Em geral, se dois literais são mutex em S_i, as ações de persistência para aqueles literais serão mutex em A_i e não precisamos indicar essa ligação de mutex.

O nível A_0 contém todas as ações que *poderiam* acontecer no estado S_0, mas, quase tão importante quanto isso, ele registra conflitos entre ações que impediriam que essas ações ocorressem juntas. As linhas em cor cinza na Figura 10.8 indicam ligações de **exclusão mútua** (ou **mutex**). Por exemplo, *Comer*(*Bolo*) é mutuamente exclusiva com a persistência de *Ter*(*Bolo*) ou ¬*Comido*(*Bolo*). Veremos em breve como as ligações de exclusão mútua são calculados.

O nível S_1 contém todos os literais que poderiam resultar da escolha de qualquer subconjunto das ações em A_0, bem como as ligações de mutex (linhas em cor cinza) indicando literais que não poderiam aparecer juntos, independentemente da escolha de ações. Por exemplo, *Ter*(*Bolo*) e *Comido*(*Bolo*) são de exclusão mútua: dependendo da escolha de ações em A_0, um ou outro, mas não ambos, poderia ser o resultado. Em outras palavras, S_1 representa um estado de crença: um conjunto de estados possíveis. Os membros desse conjunto são todos subconjuntos dos literais de tal forma que não há ligação de exclusão mútua entre quaisquer membros do subconjunto.

Continuamos, desse modo, alternando entre o nível de estado S_i e o nível de ação A_i até chegarmos a um nível em que dois níveis consecutivos são idênticos. Nesse momento, dizemos que o grafo **se nivelou**. O grafo na Figura 10.8 se nivela em S_2.

Acabaremos então com uma estrutura na qual todo nível A_i contém todas as ações que são aplicáveis em S_i, juntamente com restrições que informam que duas ações não podem ser executadas juntas no mesmo nível. Todo nível S_i contém todos os literais que poderiam resultar de qualquer escolha possível de ações em A_{i-1}, juntamente com restrições que informam quais pares de literais não são possíveis. É importante observar que o processo de construção do grafo de planejamento *não* requer a escolha entre as ações, o que exigiria uma busca combinatória. Em vez disso, ele registra apenas a impossibilidade de certas escolhas com a utilização de ligações de exclusão mútua.

Agora definimos ligações mutex para as ações e para os literais. Uma relação de mutex existe entre duas *ações* em um determinado nível, se qualquer das três condições a seguir é verdadeira:

- *Efeitos inconsistentes*: uma ação nega um efeito da outra. Por exemplo, *Comer*(*Bolo*) e a

persistência de *Ter*(*Bolo*) têm efeitos inconsistentes porque são discordantes sobre o efeito *Ter*(*Bolo*).

- *Interferência*: um dos efeitos de uma ação é a negação de uma precondição da outra. Por exemplo *Comer*(*Bolo*) interfere com a persistência de *Ter*(*Bolo*) negando sua precondição.
- *Necessidades concorrentes*: uma das precondições de uma ação é mutuamente exclusiva com uma precondição da outra. Por exemplo, *Assar*(*Bolo*) e *Comer*(*Bolo*) são mutuamente exclusivas porque competem no valor da precondição *Ter*(*Bolo*).

Uma relação de exclusão mútua existe entre dois *literais* no mesmo nível se uma é a negação da outra ou se cada par possível de ações que alcançariam os dois literais são mutuamente exclusivas. Essa condição é chamada *suporte inconsistente*. Por exemplo, *Ter*(*Bolo*) e *Comido*(*Bolo*) são de exclusão mútua em S_1 porque a única maneira de alcançar *Ter*(*Bolo*), a ação de persistência, é de exclusão mútua com a única maneira de alcançar *Comido*(*Bolo*), ou seja, *Comer*(*Bolo*). Em S_2, os dois literais não são de exclusão mútua porque existem novas maneiras de alcançá-los, como *Assar*(*Bolo*) e a persistência de *Comido*(*Bolo*), que não são de exclusão mútua.

Um grafo de planejamento é polinomial no tamanho do problema de planejamento. Para um problema de planejamento com l literais e a ações, todos os S_i não têm mais do que l nós e l^2 ligações de exclusão mútua, e cada A_i não tem mais do que $a + l$ nós (incluindo os no-ops), $(a + l)^2$ ligações de exclusão mútua e $2(al + l)$ ligações de efeito e precondição. Assim, um grafo inteiro com n níveis tem um tamanho $O(n(a+l)^2)$. O tempo para construir o grafo tem a mesma complexidade.

10.3.1 Grafos de planejamento para avaliação heurística

Uma vez construído, um grafo de planejamento é uma rica fonte de informações sobre o problema. Em primeiro lugar, se qualquer objetivo literal não aparece no nível final do grafo, o problema é insolúvel. Em segundo lugar, podemos estimar o custo de alcançar qualquer literal objetivo g_i do estado s como sendo o nível no qual g_i, aparece pela primeira vez no grafo de planejamento construído do estado inicial s. Chamamos esse custo de **custo de nível** de g_1. Na Figura 10.8, *Ter*(*Bolo*) tem custo de nível 0 e *Comido*(*Bolo*) tem custo de nível 1. É fácil mostrar (Exercício 10.10) que essas estimativas são admissíveis para os objetivos individuais. Porém, talvez a estimativa não seja sempre exata porque os grafos de planejamento permitem várias ações em cada nível, enquanto a heurística leva em conta apenas o nível, e não o número de ações. Por essa razão, é comum se utilizar um **grafo de planejamento serial** para calcular heurísticas. Um grafo serial insiste em que apenas uma ação pode realmente acontecer em qualquer passo de tempo dado; isso é feito adicionando-se ligações de exclusão mútua entre cada par de ações de não persistência. Os custos de nível extraídos de grafos seriais são muitas vezes estimativas bastante razoáveis dos custos reais.

Para estimar o custo de uma *conjunção* de objetivos, existem três abordagens simples. A heurística de **nível máximo** simplesmente considera o custo de nível máximo de qualquer dos objetivos; isso é admissível, mas não necessariamente muito preciso.

A heurística de **soma de níveis**, seguindo a hipótese de independência de subobjetivo, devolve a

soma dos custos de nível dos objetivos; isso pode ser inadmissível, mas funciona muito bem na prática para problemas amplamente decomponíveis. Ela é muito mais precisa que a heurística de número-de-objetivos-não-satisfeitos da Seção 10.2. Para o nosso problema, a soma de níveis da estimativa da heurística para o objetivo conjuntivo *Ter*(*Bolo*) ∧ *Comido*(*Bolo*) será 0 + 1 = 1, enquanto a resposta correta é 2, alcançada pelo plano [*Comer*(*Bolo*), *Assar*(*Bolo*)]. Isso não parece tão ruim. Um erro mais grave é que, se *Assar*(*Bolo*) não estiver no conjunto de ações, a estimativa ainda será 1, quando na verdade o objetivo conjuntivo seria impossível.

Finalmente, a heurística de **nível de conjunto** encontra o nível no qual todos os literais no objetivo conjuntivo aparecem no grafo de planejamento sem que nenhum par seja mutuamente exclusivo. Essa heurística fornece os valores corretos de 2 para o nosso problema original e infinito para o problema sem *Assar(Bolo*). É admissível, domina a heurística de nível máximo e funciona extremamente bem em tarefas em que há uma boa dose de interação entre os subplanos. Certamente não é perfeito; por exemplo, ignora as interações entre três ou mais literais.

Como ferramenta para gerar heurísticas precisas, podemos visualizar o grafo de planejamento como um problema relaxado que pode ser resolvido de forma eficiente. Para compreender a natureza do problema relaxado, precisamos entender exatamente o que significa um literal *g* aparecer no nível S_i no grafo de planejamento. No caso ideal, gostaríamos que ele fosse uma garantia de que existe um plano com *i* níveis de ações que atinge *g*, e também que, se *g* não aparecer, que não existe tal plano. Infelizmente, dar essa garantia é tão difícil quanto resolver o problema de planejamento original. Assim, o grafo de planejamento cuida da segunda metade da garantia (se *g* não aparecer, não existe nenhum plano), mas, se *g* aparecer, tudo o que o grafo de planejamento promete é que existe um plano que *possivelmente* atinge *g* e não tem nenhuma falha "óbvia". Uma falha óbvia é definida como uma falha que pode ser detectada considerando-se duas ações ou dois literais ao mesmo tempo — em outras palavras, examinando-se as relações de exclusão mútua. Poderia haver falhas mais sutis envolvendo três, quatro ou mais ações, mas a experiência mostrou que não vale a pena o esforço computacional para manter o controle dessas possíveis falhas. Isso é semelhante à lição aprendida a partir dos problemas de satisfação de restrições, de que frequentemente vale a pena calcular a 2-consistência antes de procurar uma solução, mas em geral é menos compensador calcular a 3-consistência ou maior.

Um exemplo de problema insolúvel, que não pode ser reconhecido como tal por um grafo de planejamento, é o problema do mundo dos blocos onde o objetivo é obter o bloco *A* sobre o *B*, *B* sobre *C* e *C* sobre *A*. Esse é um objetivo impossível; uma torre com base em cima do topo. Mas um grafo de planejamento não pode detectar porque quaisquer dois dos três subobjetivos são inatingíveis. Não há exclusões mútuas entre qualquer par de literais, apenas entre os três como um todo. Para detectar que esse problema é impossível, teríamos que fazer busca no grafo de planejamento.

10.3.2 O algoritmo Graphplan

Esta subseção mostra como extrair um plano diretamente do grafo de planejamento, em vez de apenas utilizar o grafo para fornecer uma heurística. O algoritmo GRAPHPLAN (Figura 10.9)

adiciona repetidamente um nível a um grafo de planejamento com EXPANDIR-GRAFO. Uma vez que todos os objetivos aparecem como sendo de não exclusão mútua no grafo, o GRAPHPLAN chama EXTRAIR-SOLUÇÃO para buscar um plano que resolva o problema. Se isso falhar, expande outro nível e tenta de novo, terminando em falha quando não houver razão para continuar.

função GRAPHPLAN(*problema*) **retorna** solução ou falha
    grafo ← GRAFO-DE-PLANEJAMENT0-INICIAL(*problema*)
    objetivos ← CONJUNÇÃO(*problema*.OBJETIVO)
    nãobons ← uma tabela hash vazia
    para *tl* = 0 **até** **faça**
        se *objetivos* são todos de não exclusão mútua em S_t do *grafo* **então faça**
            solução ← EXTRAIR-SOLUÇÃO(*grafo*, *objetivos*, NUMNÍVEIS(*grafo),nãobons*)
            se *solução* ≠ *falha* **então retornar** *solução*
        se *grafo e nãobons estiverem ambos nivelados* **então retornar** *falha*
        grafo ← EXPANDIR-GRAFO(*grafo*, *problema*)

Figura 10.9 O algoritmo GRAPHPLAN chama EXPANDIR-GRAFO para adicionar um nível até que seja encontrada uma solução através de EXTRAIR-SOLUÇÃO ou nenhuma solução seja possível.

Agora, vamos acompanhar a operação de GRAPHPLAN no problema do pneu sobressalente da Seção 10.10. O grafo inteiro é mostrado na Figura 10.10. A primeira linha do GRAPHPLAN inicializa o grafo de planejamento como um grafo de um único nível (S_0) representando o estado inicial. Os fluentes positivos do estado inicial da descrição do problema são mostrados, assim como os fluentes negativos relevantes. Os literais positivos que não mudam não são mostrados (como *Pneu*(*Sobressalente*)) e os literais negativos irrelevantes. O literal de objetivo *Em*(*Sobressalente*, *Eixo*) não está presente em S_0; então não precisamos da chamada a EXTRAIR-SOLUÇÃO — estamos certos de que ainda não existe nenhuma solução. Em vez disso, EXPANDIR-GRAFO adiciona em A_0 as três ações cujas precondições existem no nível S_0 (isto é, todas as ações exceto *Colocar*(*Sobressalente*, *Eixo*)), junto com ações de persistência para todos os literais em S_0. Os efeitos das ações são adicionados ao nível S_1. Depois disso, EXPANDIR-GRAFO procura por relações de exclusão mútua e as acrescenta ao grafo.

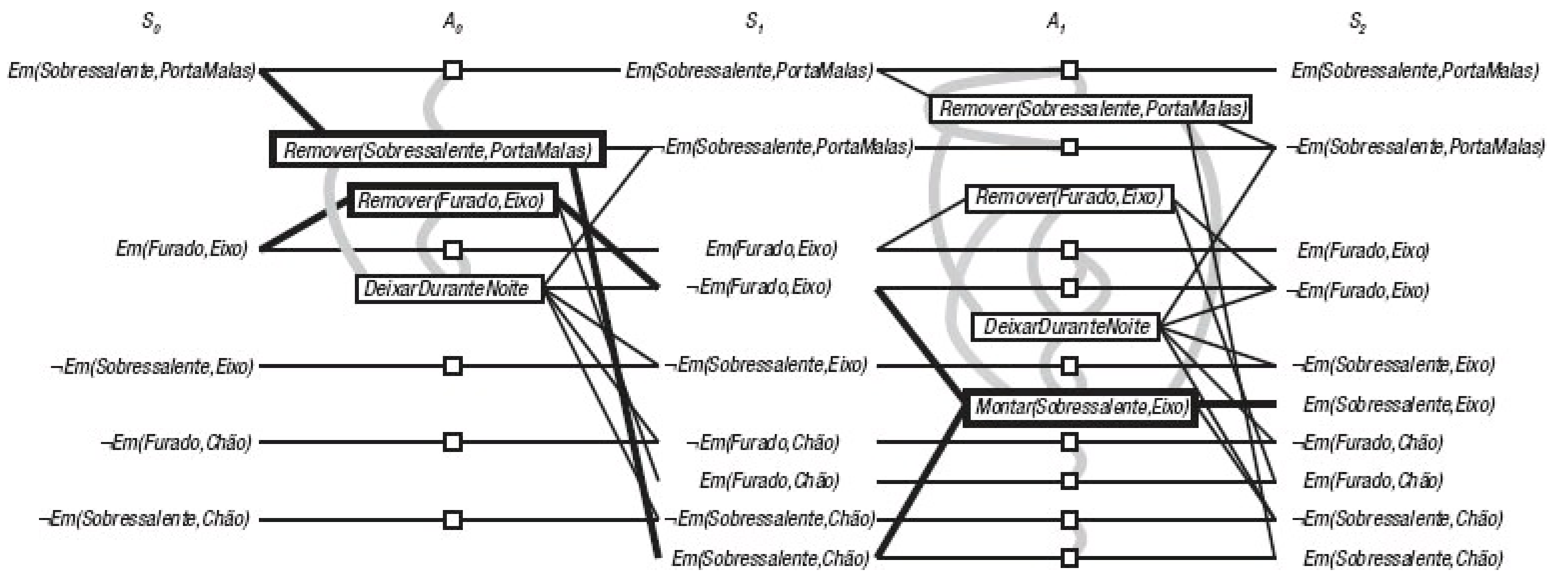


Figura 10.10 O grafo de planejamento para o problema de pneu sobressalente, depois da expansão até o nível S_2. As ligações de exclusão mútua são mostradas como linhas em cor cinza. Nem todos os mutex são mostrados porque o grafo ficaria muito confuso se mostrássemos todos eles. A solução é indicada por linhas e contornos em negrito.

Em(*Sobressalente*, *Eixo*) ainda não está presente em S_1, portanto mais uma vez não chamamos EXTRAIR-SOLUÇÃO. Chamamos EXPANDIR-GRAFO novamente, adicionando A_1 e S_1 resultando no grafo de planejamento mostrado na Figura 10.10. Agora, que temos o complemento total de ações, vale a pena examinar alguns dos exemplos de relações de exclusão mútua e suas causas:

- *Efeitos inconsistentes*: *Remover*(*Sobressalente*, *PortaMalas*) é mutex com *DeixarDuranteNoite* porque uma tem o efeito *Em*(*Sobressalente*, *Chão*) e a outra tem sua negação.
- *Interferência*: *Remover*(*Furado*, *Eixo*) é mutex com *DeixarDuranteNoite* porque uma ação tem a precondição *Em*(*Furado*, *Eixo*) e a outra tem sua negação como efeito.
- *Necessidades concorrentes*: *Colocar*(*Sobressalente*, *Eixo*) é mutex com *Remover*(*Furado*, *Eixo*), porque uma tem *Em*(*Furado*, *Eixo*) como precondição e a outra tem sua negação.
- *Suporte inconsistente*: *Em*(*Sobressalente*, *Eixo*) é mutex com *Em*(*Furado*, *Eixo*) em S_2 porque a única maneira de alcançar *Em*(*Sobressalente*, *Eixo*) é através de *Colocar*(*Sobressalente*, *Eixo*), e essa ação é mutex com a ação de persistência que é o único modo de alcançar *Em*(*Furado*, *Eixo*). Desse modo, as relações mutex detectam o conflito imediato que surge a partir da tentativa de colocar dois objetos no mesmo lugar ao mesmo tempo.

Dessa vez, quando voltamos ao início da repetição, todos os literais do objetivo estão presentes em S_2, e nenhum deles é mutex com qualquer outro. Isso significa que poderia existir uma solução, e EXTRAIR-SOLUÇÃO tentará descobri-la. Podemos formular, EXTRAIR-SOLUÇÃO como um problema de satisfação de restrição booleano (CSP) onde as variáveis são as ações em cada nível, os valores para cada variável estão *dentro* ou *fora* do plano, as restrições são as exclusões mútuas e há necessidade de satisfazer cada objetivo e precondição.

Podemos definir alternativamente EXTRAIR-SOLUÇÃO como um problema de busca para trás, no qual cada estado na busca contém um ponteiro para um nível no grafo de planejamento e um conjunto de objetivos não satisfeitos. Definimos esse problema de busca assim:

- O estado inicial é o último nível do grafo de planejamento, S_n, juntamente com o conjunto de objetivos do problema de planejamento.
- As ações disponíveis em um estado no nível S_i se destinam a selecionar qualquer subconjunto livre de conflitos das ações em A_{i-1} cujos efeitos cobrem os objetivos do estado. O estado resultante tem nível S_{i-1} e tem como seu conjunto de objetivos as precondições correspondentes ao conjunto de ações selecionado. Por "livre de conflitos" queremos dizer um conjunto de ações tais que duas quaisquer dessas ações não sejam mutuamente exclusivas e que duas quaisquer de suas precondições não sejam mutuamente exclusivas.
- O objetivo é alcançar um estado no nível S_0 tal que todos os objetivos sejam satisfeitos.
- O custo de cada ação é 1.

Para esse problema específico, começamos em S_2 com o objetivo *Em*(*Sobressalente*, *Eixo*). A única escolha que temos para alcançar o conjunto de objetivos é *Colocar*(*Sobressalente*, *Eixo*). Isso nos leva a um estado de busca em S_1 com os objetivos *Em*(*Sobressalente*, *Chão*) e ¬*Em*(*Furado*, *Eixo*). O primeiro só pode ser alcançado por *Remover*(*Sobressalente*, *PortaMalas*), e o último por *Remover*(*Furado*, *Eixo*) ou *DeixarDuranteNoite*. Porém, *Deixar-DuranteNoite* é mutex com *Remover*(*Sobressalente*, *PortaMalas*); então, a única solução é escolher *Remover*(*Sobressalente*, *PortaMalas*) e *Remover*(*Furado*, *Eixo*). Isso nos leva a um estado de busca em S_0 com os objetivos *Em*(*Sobressalente*, *PortaMalas*) e *Em*(*Furado*, *Eixo*). Ambos estão presentes no estado, e então temos uma solução: as ações *Remover*(*Sobressalente*, *PortaMalas*) e *Remover*(*Furado*, *Eixo*) no nível A_0, seguidas por *Colocar*(*Sobressalente*,*Eixo*) em A_1.

No caso em que EXTRAIR-SOLUÇÃO falhar em encontrar uma solução para um conjunto de objetivos em determinado nível, gravamos o par (*nível, objetivos*), como um **não-bom**, assim como fizemos na restrição de aprendizagem para CSPs. Sempre que EXTRAIR- SOLUÇÃO for chamado novamente com o mesmo nível e objetivos, podemos encontrar o não-bom registrado e devolver imediatamente a falha em vez de buscar novamente. Vemos logo que não-bons são também utilizados no teste de término.

Sabemos que o planejamento é PSPACE-completo e que a construção do grafo de planejamento demora um tempo polinomial; assim, a extração da solução é intratável no pior caso. Por conseguinte, precisaremos de alguma orientação de heurística para escolher entre ações durante a busca para trás. Uma abordagem que funciona bem na prática é um algoritmo guloso baseado no custo de nível dos literais. Para qualquer conjunto de objetivos, prosseguimos na seguinte ordem:

1. Selecionar primeiro o literal com o custo de nível mais alto.
2. Para atingir esse literal, escolha ações com precondições fáceis. Isto é, escolher uma ação tal que a soma (ou o máximo) dos custos de nível de suas precondições seja o menor possível.

10.3.3 Término do GRAPHPLAN

Até agora, evitamos discutir a questão do término. Aqui nós mostramos que o GRAPHPLAN, na

verdade, terminará e devolverá falha quando não houver solução.

A primeira coisa a entender é por que não podemos parar de expandir o grafo logo que ele nivela. Considere um domínio de carga aérea com um avião e n peças de carga no aeroporto A, todos tendo o aeroporto B como destino. Nessa versão do problema, apenas uma peça da carga pode caber no avião de cada vez. O grafo vai nivelar em 4, refletindo o fato de que, para qualquer peça única de carga, podemos carregá-la, transportá-la e descarregá-la no local de destino em três passos. Mas isso não significa que uma solução possa ser extraída do grafo no nível 4; de fato, uma solução exigirá $4n - 1$ passos: para cada peça de carga que carregamos, transportamos e descarregamos, e para todas, exceto a última peça, teremos que voltar ao aeroporto A para obter a próxima peça.

Quanto tempo temos que continuar expandindo depois que o grafo for nivelado? Se a função EXTRAIR-SOLUÇÃO falhar em encontrar uma solução, deve haver pelo menos um conjunto de objetivos que não foram atingidos e foram marcados como não-bons. Então, se é possível que haja poucos não-bons no próximo nível, devemos continuar. Assim que o grafo em si e os não-bons tiverem nivelados, sem solução encontrada, podemos terminar com falha porque não há possibilidade de uma alteração posterior que poderia adicionar uma solução.

Agora tudo o que temos a fazer é provar que o grafo e os não-bons sempre vão nivelar. A chave para essa prova é que certas propriedades de grafos de planejamento são monotonicamente crescentes ou decrescentes. "X aumenta monotonicamente" significa que o conjunto de valores de X no nível $i + 1$é um superconjunto (não necessariamente próprio) do conjunto do nível i. As propriedades são as seguintes:

- *Literais aumentam monotonicamente*: Depois que um literal aparecer em um dado nível, ele aparecerá em todos os níveis subsequentes. Isso ocorre devido às ações de persistência; uma vez que um literal surge, as ações de persistência o fazem permanecer para sempre.
- *Ações aumentam monotonicamente*: Depois que uma ação aparecer em dado nível, ela aparecerá em todos os níveis subsequentes. Essa é uma consequência do crescimento monotônico dos literais; se as precondições de uma ação aparecerem em um nível, elas aparecerão em níveis subsequentes, portanto a ação também aparecerá.
- *As exclusões mútuas diminuem monotonicamente*: Se duas ações são de exclusão mútua em um dado nível A_i, então elas também são de exclusão mútua em todos os níveis *anteriores* em que ambas aparecem. O mesmo é válido no caso de exclusões mútuas entre literais. Nem sempre pode parecer desse modo nas figuras porque as figuras têm uma simplificação: elas não exibem os literais que não podem ser válidos no nível S_i nem as ações que não podem ser executadas no nível A_i. Podemos observar que "as exclusões mútuas diminuem monotonicamente" é verdadeiro se considerarmos que esses literais e essas ações invisíveis são de exclusão mútua com todos os outros.

  A prova pode ser tratada por casos: se as ações A e B são mutex no nível A_i, isso ocorre devido a um dos três tipos de mutex. Os dois primeiros, efeitos inconsistentes e interferência, são propriedades das próprias ações; assim, se as ações forem mutex em A_i, elas serão mutex em qualquer nível. O terceiro caso, necessidades concorrentes, depende das condições no nível S_i: esse nível deve conter uma precondição de A que seja de mutex com uma precondição de B.

Agora, essas duas precondições podem ser mutex se forem negações uma da outra (em cujo caso seriam mutex em todos os níveis) ou se todas as ações para alcançar uma delas são mutex com todas as ações para alcançar a outra. Porém, já sabemos que as ações disponíveis estão aumentando monotonicamente; então, por indução, as exclusões mútuas devem estar diminuindo.

- *Não-bons diminuem monotonicamente*: Se um conjunto de objetivos não for atingível em um determinado nível, eles não são atingíveis em qualquer nível *anterior*. A prova é por contradição: se eles forem atingíveis em algum nível *anterior*, então poderemos apenas adicionar ações de persistência para torná-los atingíveis em um nível subsequente.

Porque as ações e os literais aumentam monotonicamente e também pelo fato de haver apenas um número finito de ações e literais, deve haver um nível que tem o mesmo número de ações e literais como no nível anterior. Deve ocorrer um nível que tenha o mesmo número de exclusão mútua e de não-bons, como no nível anterior, pois os mutuamente exclusivos e os não-bons diminuem, e nunca pode haver menos que zero de exclusão mútua ou não-bons. Uma vez que um grafo atinja esse estado, se um dos objetivos estiver ausente ou for de exclusão mútua com outro objetivo, podemos parar o algoritmo GRAPHPLAN e devolver falha. Isso conclui um esboço da prova; para mais detalhes, consulte Ghallab *et al*. (2004).

10.4 OUTRAS ABORDAGENS CLÁSSICAS DE PLANEJAMENTO

Atualmente, as abordagens mais populares e eficazes para o planejamento totalmente automatizado são:

- Tradução para um problema de satisfatibilidade booleana (SAT)
- Busca em espaço de estados para a frente com heurísticas bem construídas (Seção 10.2)
- Busca utilizando um grafo de planejamento (Seção 10.3)

Essas três abordagens não foram os únicas tentativas em 40 anos de história de planejamento automatizado. A Figura 10.11 mostra alguns dos sistemas de topo nas *International Planning Competitions*, que foram realizados a cada ano, desde 1998. Nesta seção, primeiro descreveremos a tradução para um problema de satisfatibilidade e depois descreveremos três outras abordagens influentes: o planejamento como dedução lógica de primeira ordem, como satisfação de restrição e como refinamento do plano.

Ano	Trilha	Sistemas Winning (abordagens)
2008 2008	Ótimo Satisfatório	GAMER (verificação do modelo, busca bidirecional) LAMA (busca rápida decrescente com heurística FF)
2006 2006	Ótimo Satisfatório	SATPLAN, MAXPLAN (satisfatibilidade booleana) SGPLAN (busca para a frente; partições em subproblemas independentes)
2004 2004	Ótimo Satisfatório	SATPLAN (satisfatibilidade booleana) RÁPIDO EM DIAGONAL DECRESCENTE (busca para a frente com

		grafo causal)
2002 2002	Automatizado Codificado à mão	GLP (busca local, grafos de planejamento convertido para PSRs) TLPLAN (ação lógica temporal com regras de controle de busca para a frente)
2000 2000	Automatizado Codificado à mão	FF (busca para a frente) TALPLANNER (ação lógica temporal com regras de controle de busca para a frente)
1998	Automatizado	IPP (grafos de planejamento); HSP (busca para a frente)

Figura 10.11 Alguns dos sistemas de alto desempenho na *International Planning Competition*. A cada ano há várias trilhas: "ótimo" significa que os planejadores devem produzir o plano mais curto possível, enquanto "satisfatório" significa que soluções não ótimas são aceitas. "Codificado à mão" significa que são permitidas heurísticas de domínio específico; "automatizado" significa que não são.

10.4.1 Planejamento clássico como satisfatibilidade booleana

Na Seção 7.7.4, vimos como o SATPLAN resolve os problemas de planejamento que são expressos em lógica proposicional. Aqui mostraremos como traduzir uma descrição PDDL em uma forma que pode ser processada por SATPLAN. A tradução é composta de uma série de etapas simples:

- Proposicionalizar as ações: substituir cada esquema de ação com um conjunto de ações instanciadas formadas pela substituição de constantes para cada uma das variáveis. Essas ações instanciadas não são parte da tradução, mas serão utilizadas nas etapas subsequentes.
- Definir o estado inicial: declarar F^0 para cada fluente F no estado inicial do problema, e $\neg F$ para cada fluente não mencionado no estado inicial.
- Proposicionalizar o objetivo: para cada variável no objetivo, substituir os literais que contêm uma variável com uma disjunção sobre constantes. Por exemplo, o objetivo de ter o bloco A sobre outro bloco, $Sobre(A, x) \wedge Bloco(x)$ em um mundo com objetos A, B e C, seria substituído pelo objetivo

$$(Sobre(A, A) \wedge Bloco(A)) \vee (Sobre(A, B) \wedge Bloco(B)) \vee (Sobre(A, C) \wedge Bloco(C)).$$

- Adicionar axiomas de estado sucessor: para cada fluente F, adicionar um axioma da forma

$$F^{t+1} \Leftrightarrow AçãoCausaF^t \vee (F^t \wedge \neg AçãoCausaNãoF^t),$$

  onde *AçãoCausaF* é uma disjunção de todas as ações instanciadas, que tem F em sua lista de adição e *AçãoCausaNãoF* é uma disjunção de todas as ações instanciadas, que tem F em sua lista de exclusão.
- Adicionar axiomas de precondição: para cada ação fundamentada A, adicionar o axioma $A^t \Rightarrow$ PRE $(A)^t$, isto é, se for tomada uma ação no tempo t, é porque as precondições eram verdadeiras.

- Adicionar axiomas de exclusão de ação: informa que toda ação é distinta de qualquer outra ação.

A tradução resultante estará na forma em que possamos passar ao SATPLAN para encontrar uma solução.

10.4.2 Planejamento como dedução na lógica de primeira ordem: cálculo de situação

PDDL é uma linguagem que equilibra cuidadosamente a expressividade da linguagem com a complexidade dos algoritmos que a operam. Mas alguns problemas permanecem difíceis de expressar em PDDL. Por exemplo, não podemos expressar o objetivo "mover toda a carga de *A* para *B*, independentemente de quantas peças de carga existam" em PDDL, mas podemos fazê-lo na lógica de primeira ordem usando um quantificador universal. Da mesma forma, a lógica de primeira ordem pode expressar concisamente restrições globais, tais como "não mais que quatro robôs podem estar no mesmo lugar, ao mesmo tempo". PDDL só pode dizer isso com precondições repetitivas em cada ação possível que envolve um movimento.

A representação em lógica proposicional de problemas de planejamento também tem limitações, tal como o fato de que a noção de tempo está diretamente ligada aos fluentes. Por exemplo, Sul^2 significa "o agente está voltado para o sul no tempo 2". Com essa representação, não há nenhuma maneira de dizer "o agente estaria voltado para o sul no tempo 2 se executasse uma curva à direita no tempo 1, caso contrário estaria voltado para leste". A lógica de primeira ordem nos permite contornar essa limitação substituindo a noção do tempo linear com uma noção de *situações* de ramificação, usando uma representação chamada **cálculo de situações** que funciona assim:

- O estado inicial é chamado de **situação**. Se *s* for uma situação e *a* for uma ação, RESULTADO(*s*, *a*) também será uma situação. Não há outras situações. Assim, uma situação corresponde a uma sequência ou histórico de ações. Pode-se também pensar em uma situação como o resultado da aplicação de ações, mas observe que duas situações serão as mesmas apenas se o seu início e as ações forem os mesmos: (RESULTADO(s, a) = RESULTADO(s', a')) $\Leftrightarrow$ ($s = s' \land a = a'$). Alguns exemplos de ações e situações são mostrados na Figura 10.12.

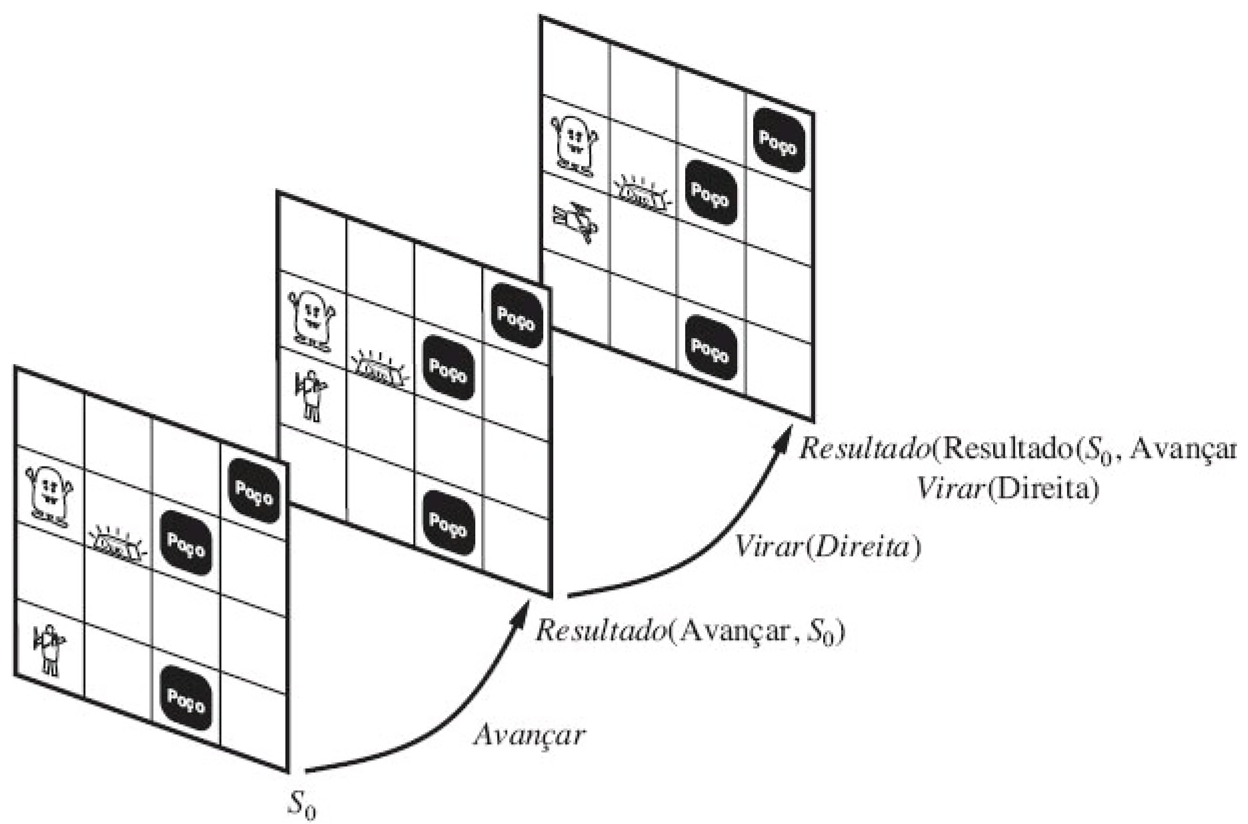


Figura 10.12 Situações como resultados de ações no mundo de wumpus.

- A função ou relação que pode variar de uma situação para a próxima é um **fluente**. Por convenção, a situação *s* é sempre o último argumento para o fluente, por exemplo, *Em*(*x, l, s*) é um fluente relacional que é verdadeiro quando o objeto *x* estiver na localização *l* e na situação *s*, e *Localização* é um fluente funcional tal que *Localização*(*x, s*) = *l* é verdadeiro nas mesmas situações que *Em*(*x, l, s*).
- Cada uma das precondições da ação é descrita com um **axioma de possibilidade** que informa quando a ação pode ser tomada. Ele tem a forma $\Phi(s) \Rightarrow Possível(a, s)$ onde $\Phi(s)$ é alguma fórmula envolvendo *s* que descreve as precondições. Um exemplo do mundo de wumpus diz que é possível atirar se o agente estiver vivo e tiver uma flecha:

$$Vivo(Agente, s) \land Tem(Agente, Flecha, s) \Rightarrow Possível(Atirar, s).$$

- Cada fluente é descrito com um **axioma de estado sucessor** que diz o que acontece com o fluente, dependendo da ação que for realizada. Esta é semelhante à abordagem que adotamos para a lógica proposicional. O axioma tem a forma

Ação é possível ⇒
(Fluente é verdadeiro no estado resultado ⇔ *Efeito da ação tomada é verdadeira*
∨ *Era verdadeiro antes e a ação não o alterou).*

Por exemplo, o axioma do fluente relacional *Segurar* diz que o agente está segurando algum ouro *g* depois de executar uma ação possível se e somente se a ação era *Agarrar* de *g* ou se o agente já estava segurando *g* e a ação não o solta:

$$\text{Possível}(a, s) \Rightarrow$$
$$(Segurar(Agente, g, Resultad\ (a, s)) \Leftrightarrow$$
$$a = Agarrar(g) \vee (Segurar(Agente, g, s) \wedge a \neq Soltar(\mathrm{g}))).$$

- Precisamos de **axiomas de ação única** para que o agente possa deduzir que, por exemplo, $a \neq Soltar(g)$. Para cada par de nomes distintos de ação A_i e A_j temos um axioma que informa que as ações são diferentes:

$$A_i\ (x,\ldots) \neq A_j\ (y,\ldots)$$

e para cada nome da ação A_i temos um axioma que informa que dois usos desse nome da ação são iguais se e somente se todos os seus argumentos forem iguais:

$$A_i\ (x_1,\ldots, x_n) = A_i\ (y_1,\ldots, Y_n)\ \Leftrightarrow x_1 = y_1 \wedge \ldots \wedge x_n = y_n.$$

- A solução é uma situação (e, portanto, uma sequência de ações) que satisfaz o objetivo.

Trabalhos em cálculo de situações fizeram muito para definir a semântica formal de planejamento e para abrir novas áreas de investigação. Mas até agora não houve um programa prático de planejamento em larga escala com base em dedução lógica sobre o cálculo de situações. Isso é em parte devido à dificuldade de se fazer inferência eficiente em FOL, mas é principalmente porque o campo ainda não desenvolveu heurística eficazes de planejamento com cálculo de situações.

10.4.3 Planejamento como satisfação de restrição

Vimos que a satisfação de restrição tem muito em comum com a satisfatibilidade booleana e que as técnicas de CSP são eficazes para problemas de escalonamento, de modo que não é surpreendente que seja possível codificar um problema de planejamento limitado (ou seja, o problema de encontrar um plano de comprimento *k*) como um problema de satisfação de restrição (CSP). A codificação é similar à codificação de um problema SAT (Seção 10.4.1), com uma simplificação importante: em cada etapa precisamos apenas de uma única variável, $Ação^t$, cujo domínio é o conjunto de ações possíveis. Não é necessário mais que uma variável para cada ação, e não precisamos dos axiomas de exclusão de ação. É possível também codificar um grafo de planejamento em um CSP. Essa é a abordagem do GP-PSR (Do e Kambhampati, 2003).

10.4.4 Planejamento como refinamento de planos parcialmente ordenados

Todas as abordagens que vimos até agora constroem planos *totalmente ordenados* consistindo em uma sequência de ações totalmente ordenada. Essa representação ignora o fato de que muitos subproblemas são independentes. Uma solução para um problema de carga aérea consiste em uma sequência de ações totalmente ordenadas, mas se 30 peças forem carregadas em um avião em um aeroporto e 50 peças forem carregadas em outro avião em outro aeroporto, parece inútil chegar com

uma ordenação completa de 80 ações de carga; os dois subconjuntos de ações deveriam ser pensados de forma independente.

Uma alternativa é representar planos como estruturas *parcialmente ordenadas*: um plano é um conjunto de ações e um conjunto de restrições da forma *Antes*(a_i, a_j) informando que uma ação ocorre antes da outra. Na parte inferior da Figura 10.13, vemos um plano parcialmente ordenado que é uma solução para o problema do pneu sobressalente. As ações são caixas e as restrições de ordenação são setas. Observe que *Remover*(*Sobressalente, Porta-malas*) e *Remover* (*Furado, Eixo*) podem ser feitos em qualquer ordem, desde que sejam concluídos antes da ação *Colocar*(*Sobressalente, Eixo*).

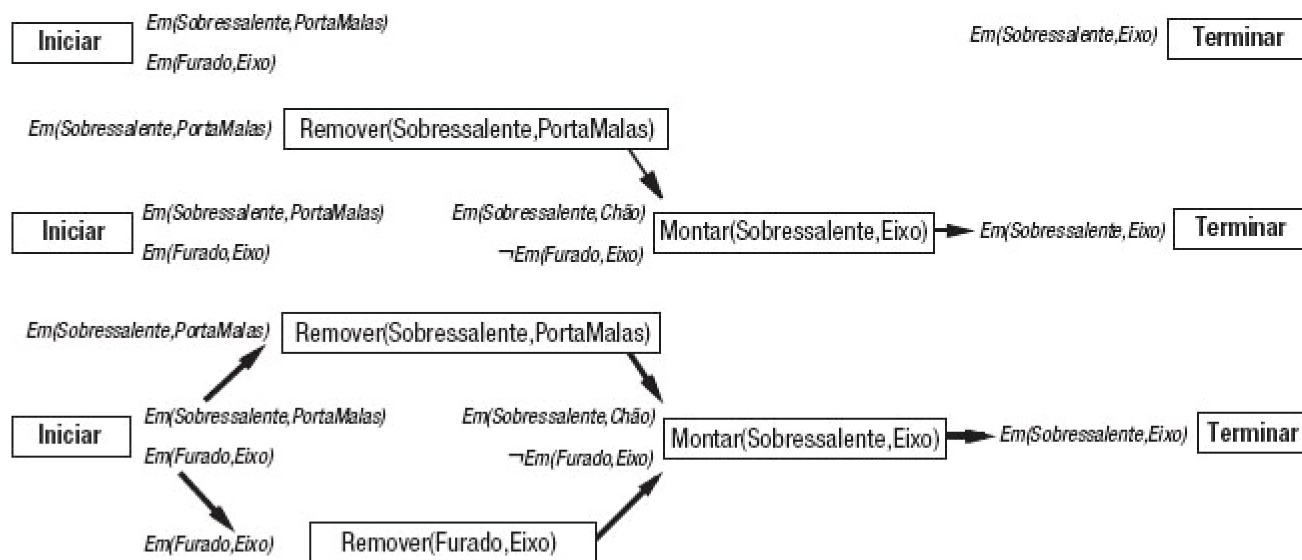


Figura 10.13 (a) O problema do pneu descrito como um plano vazio. (b) Um plano incompleto parcialmente ordenado do problema do pneu. As caixas representam as ações, e as setas indicam que uma ação deve ocorrer antes da outra. (c) Uma solução completa parcialmente ordenada.

Os planos parcialmente ordenados são criados por uma *busca através do espaço dos planos*, em vez de através do espaço de estados. Começamos com o plano vazio consistindo apenas no estado inicial e no objetivo, sem ações entre eles, como na parte superior da Figura 10.13. O procedimento de busca, procura então por uma **falha** no plano e faz uma adição ao plano para corrigir a falha (ou, se não puder ser feita nenhuma correção, a busca retrocede e tenta outra coisa). A falha é algo que impede o plano parcial de ser uma solução. Por exemplo, uma falha no plano vazio é que nenhuma ação atinge *Em*(*Sobressalente*, *Eixo*). Uma forma de corrigir a falha é inserir no plano a ação *Colocar*(*Sobressalente*, *Eixo*). Isso certamente introduz algumas falhas novas: as precondições da nova ação não são alcançadas. A busca continua fazendo adições ao plano (retrocedendo se necessário) até que todas as falhas sejam resolvidas, como na parte inferior da Figura 10.13. A cada etapa, fazemos o **menor compromisso** possível para corrigir a falha. Por exemplo, ao adicionar a ação *Remover*(*Sobressalente, Porta-malas*) temos que nos comprometer que ocorra antes de *Colocar*(*Sobressalente, Eixo*), mas não nos comprometemos em colocá-lo antes ou depois de outras ações. Se houvesse uma variável no esquema da ação que pudesse ser deixada ilimitada, deixaríamos.

Nas décadas de 1980 e 1990, o planejamento de ordem parcial era visto como a melhor maneira de lidar com problemas de planejamento com subproblemas independentes porque, acima de tudo, era a única abordagem que representava explicitamente ramificações independentes de um plano. Por

outro lado, tinha a desvantagem de não ter uma representação explícita dos estados no modelo de transição de estados. Isso tornava alguns cálculos complicados. Em 2000, os planejadores de busca para a frente desenvolveram excelentes heurísticas que lhes permitiam descobrir eficientemente subproblemas independentes para os quais o planejamento de ordem parcial foi projetado. Como resultado, os planejadores de ordem parcial não são competitivos em problemas de planejamento clássico totalmente automatizado.

No entanto, o planejamento de ordem parcial continua a ser uma parte importante da área. Para algumas tarefas específicas, como operações de escalonamento, o planejamento de ordem parcial com heurística de domínio específico é a tecnologia a ser escolhida. Muitos desses sistemas utilizam bibliotecas de planos de alto nível, como descrito na Seção 11.2. O planejamento de ordem parcial é também frequentemente utilizado em domínios em que seja importante que os seres humanos compreendam os planos. Os planejadores de ordem parcial geram os planos operacionais para naves espaciais e robôs em Marte, que são então verificados por operadores humanos antes de serem carregados nos veículos para execução. A abordagem de refinamento do plano torna mais fácil para os seres humanos compreender o que os algoritmos de planejamento estão fazendo e verificar se eles estão corretos.

10.5 ANÁLISE DAS ABORDAGENS DE PLANEJAMENTO

Planejamento combina as duas principais áreas de IA que estudamos até agora: *busca* e *lógica*. Um planejador pode ser visualizado como um programa que procura por uma solução ou como um programa que demonstra (construtivamente) a existência de uma solução. O cruzamento de ideias das duas áreas levou a aperfeiçoamentos de desempenho que totalizaram várias ordens de magnitude na última década e também a um uso crescente de planejadores em aplicações industriais. Infelizmente, ainda não temos uma compreensão clara de quais técnicas funcionam melhor em cada um dos tipos de problemas. É bem possível que venham a emergir novas técnicas que vão superar os métodos existentes.

Planejamento é antes de tudo um exercício de controle da explosão combinatória. Se houver n proposições em um domínio, existirão 2^n estados. Como vimos, o planejamento é PSPACE-difícil. Contra esse pessimismo, a identificação de subproblemas independentes pode ser uma arma poderosa. No melhor caso — capacidade de decomposição integral do problema —, temos aumento de desempenho exponencial. No entanto, a decomposição é destruída por interações negativas entre as ações.

O GRAPHPLAN registra exclusões mútuas para apontar onde estão as dificuldades de interação. O SATPLAN representa um conjunto semelhante de relações de excluvidade mútua, mas faz isso usando a forma geral FNC, em vez de uma estrutura de dados específica. A busca para a frente aborda o problema heuristicamente tentando encontrar padrões (subconjuntos de proposições) que cubram os subproblemas independentes. Uma vez que essa abordagem é heurística, ela pode funcionar mesmo quando os subproblemas não são completamente independentes.

Às vezes é possível resolver um problema de modo eficiente reconhecendo que interações negativas podem ser eliminadas. Dizemos que um problema tem **subobjetivos serializáveis** se existe

uma ordem de subobjetivos tal que o planejador pode alcançá-los nessa ordem, sem ter de desfazer quaisquer dos subobjetivos alcançados anteriormente. Por exemplo, no mundo de blocos, se o objetivo é construir uma torre (por exemplo, *A* sobre *B*, que por sua vez está sobre *C*, que por sua vez está sobre a *Mesa,* como na Figura 10.4), então os subobjetivos são serializáveis de baixo para cima: se alcançarmos primeiro *C* sobre *Mesa*, nunca teremos de desfazê-lo enquanto estivermos alcançando os outros subobjetivos. Um planejador que utiliza a abordagem de baixo para cima pode resolver qualquer problema no domínio do mundo de blocos sem retrocesso (embora nem sempre possa descobrir o plano mais curto).

Como um exemplo mais complexo, para o planejador de Remot Agent que comandou a espaçonave Deep Space One da NASA, determinou-se que as proposições envolvidas no comando de uma espaçonave são serializáveis. Isso talvez não seja surpreendente porque uma espaçonave é projetada por seus engenheiros para ser tão fácil quanto possível de controlar (sujeita a outras restrições). Tirando proveito da ordenação serializada de objetivos, o planejador do Remot Agent foi capaz de eliminar a maior parte da busca. Isso significa que ele foi rápido o bastante para controlar a espaçonave em tempo real, algo anteriormente considerado impossível.

Não há dúvida de que planejadores como GRAPHPLAN, SATPLAN e FF produziram avanços no campo do planejamento, tanto por elevarem o nível de desempenho de sistemas de planejamento quanto por esclarecerem as questões combinatórias e de representação envolvidas, e pelo desenvolvimento de heurísticas úteis. No entanto, questiona-se até onde essas técnicas em termos de tamanho de problema. Parece provável que o progresso para problemas maiores não possa contar apenas com representações fatorada e proposicional, e vai exigir algum tipo de síntese de representações de primeira ordem e hierárquica com as heurísticas eficientes em uso atualmente.

10.6 RESUMO

Neste capítulo, definimos o problema do planejamento em ambientes determinísticos completamente observáveis e estáticos. Descrevemos as representações PDDL usadas em problemas de planejamento e diversas abordagens algorítmicas para resolvê-los. Os pontos a serem lembrados são:

- Os sistemas de planejamento são algoritmos de resolução de problemas que operam sobre representações explícitas proposicionais (ou de primeira ordem) de estados e ações. Essas representações tornam possível a derivação de heurísticas efetivas e o desenvolvimento de algoritmos poderosos e flexíveis para resolução de problemas.
- A PDDL, a linguagem de definição do domínio do planejamento, descreve os estados inicial e objetivo como combinações de literais, e ações em termos de suas precondições e efeitos.
- A busca no espaço de estados pode operar no sentido para a frente (**progressão**) ou no sentido para trás (**regressão**). Podem ser derivadas heurísticas efetivas adotando-se uma hipótese de independência de subobjetivos e empregando-se vários relaxamentos do problema de planejamento.
- Um **grafo de planejamento** pode ser construído de forma incremental, partindo-se do estado

inicial. Cada camada contém um superconjunto de todos os literais ou ações que poderiam ocorrer nesse passo de tempo e codifica relações de exclusão mútua (**mutex**) entre literais ou ações que não podem ocorrer concomitantemente. Os grafos de planejamento geram heurísticas úteis para planejadores de espaço de estados e de ordem parcial e podem ser empregados diretamente no algoritmo GRAPHPLAN.
- Outras abordagens incluem a dedução de primeira ordem sobre axiomas de cálculo de situações; codificação de um problema de planejamento como problema de satisfatibilidade booleana ou como um problema de satisfação de restrição; e busca explícita através do espaço de planos parcialmente ordenados.
- Cada uma das principais abordagens de planejamento tem seus pontos fortes, e ainda não existe nenhum consenso sobre qual delas é a melhor. A competição e a fertilização cruzada entre as abordagens resultaram em ganhos significativos em eficiência para sistemas de planejamento.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O planejamento da IA surgiu de investigações em busca no espaço de estados, prova de teoremas e teoria de controle, e das necessidades práticas da robótica, escalonamento e de outros domínios. STRIPS (Fikes e Nilsson, 1971), o primeiro sistema de planejamento importante, ilustra a interação dessas influências. STRIPS foi projetado como o componente de planejamento do software para o projeto do robô Shakey na SRI. Sua estrutura de controle global foi modelada sobre a do GPS, o General Problem Solver (Newell e Simon, 1961), um sistema de busca no espaço de estados que utilizava a análise de meios-fins. Bylander (1992) mostra o planejamento simples de STRIPS para se tornar PSPACE-completa. Fikes e Nilsson (1993) apresentam uma retrospectiva histórica sobre o projeto de STRIPS e seu relacionamento com esforços de planejamento mais recentes.

A linguagem de representação usada por STRIPS tem sido muito mais influente que sua abordagem algorítmica; o que chamamos de linguagem "clássica" se aproxima do que STRIPS usou. A *Action Description Language*, ou ADL (Pednault, 1986), relaxou algumas das restrições da linguagem STRIPS e tornou possível codificar problemas mais realistas. Nebel (2000) explora esquemas para compilar a ADL em STRIPS. A Problem Domain Description Language ou PDDL (Ghallab *et al*., 1998) foi introduzida como uma sintaxe padronizada e traduzível pelo computador para representar problemas de planejamento e tem sido usada como a linguagem-padrão da competição internacional de planejamento desde 1998. Houve várias extensões; a versão mais recente, PDDL 3.0, inclui restrições de plano e preferências (Gerevini e Long, 2005).

Os planejadores do início da década de 1970 em geral consideravam sequências de ações totalmente ordenadas. A decomposição de problemas foi realizada pela computação de um subplano para cada subobjetivo e depois pelo encadeamento dos subplanos em alguma ordem. Essa abordagem, chamada **planejamento linear** por Sacerdoti (1975), logo se mostrou incompleta. Ela não é capaz de resolver alguns problemas muito simples, como a anomalia de Sussman (veja o Exercício 10.7), encontrada por Allen Brown durante a experimentação com o sistema HACKER (Sussman, 1975). Um planejador completo deve permitir a **intercalação** de ações de diferentes subplanos dentro de uma única sequência. A noção de subobjetivos serializáveis (Korf, 1987)

corresponde exatamente ao conjunto de problemas para os quais os planejadores não intercalados são completos.

Uma solução para o problema de intercalação foi o planejamento por regressão de objetivo, uma técnica em que os passos de um plano totalmente ordenado são reordenados para evitar conflitos entre subobjetivos. Essa técnica foi introduzida por Waldinger (1975) e também foi usada pelo WARPLAN de Warren (1974). O WARPLAN também é notável pelo fato de ter sido o primeiro planejador escrito em uma linguagem de programação lógica (Prolog) e é um dos melhores exemplos da notável economia que às vezes se pode obter pelo uso da programação de lógica: o WARPLAN tem apenas 100 linhas de código, uma pequena fração do tamanho de planejadores comparáveis da época.

As ideias subjacentes ao planejamento de ordem parcial incluem a detecção de conflitos (Tate, 1975a) e a proteção de condições alcançadas de interferências (Sussman, 1975). A construção de planos parcialmente ordenados (chamados de **redes de tarefas**) teve como pioneiros o planejador NOAH (Sacerdoti, 1975, 1977) e o sistema NONLIN de Tate (1975b, 1977).

O planejamento de ordem parcial dominou os 20 anos de pesquisa seguintes, mas a primeira proposta formal clara foi o TWEAK (Chapman, 1987), um planejador que era simples o suficiente para permitir provas de completeza e intratabilidade (NP-dificuldade e irresolubilidade) de várias formulações do problema de planejamento. O trabalho de Chapman levou a uma descrição simples de um planejador de ordem parcial completo (McAllester e Rosenblitt, 1991), em seguida para as implementações amplamente distribuídas SNLP (Soderland e Weld, 1991) e UCPOP (Penberthy e Weld, 1992). O planejamento de ordem parcial caiu em desuso na década de 1990 à medida que surgiram métodos mais rápidos. Nguyen e Kambhampati (2001) sugerem que uma reconsideração é merecida: com heurísticas exatas derivadas de um grafo de planejamento, seu planejador REPOP escala muito melhor que o GRAPHPLAN em domínios paralelizáveis e é competitivo com os mais rápidos planejadores em espaço de estados.

O ressurgimento do interesse no planejamento nos espaços de estados teve como pioneiro o programa UNPOP de Drew McDermott (1996), que foi o primeiro a sugerir uma heurística de ignorar as listas de exclusão. O nome UNPOP foi uma reação à concentração exagerada no planejamento de ordem parcial existente na época; McDermott suspeitava que outras abordagens não estavam obtendo a atenção que mereciam. O *Heuristic Search Planner* (HSP) de Bonet e Geffner e seus derivados posteriores (Bonet e Geffner, 1999; Haslum *et al*., 2005; Haslum, 2006) foram os primeiros a tornar prática a busca em espaços de estados para grandes problemas de planejamento. HSP busca na direção para a frente, enquanto HSPR busca para trás (Bonet e Geffner, 1999). O mais bem-sucedido buscador em espaço de estados até hoje é o FF (Hoffmann, 2001; Hoffmann e Nebel, 2001; Hoffmann, 2005), vencedor da competição de planejamento AIPS 2000. O FASTDOWNWARD (Helmert, 2006) é um planejador de busca em espaço de estados para a frente que pré-processa os esquemas de ação em uma representação alternativa que torna algumas das restrições mais explícitas. O FASTDOWNWARD (Helmert e Richter, 2004; Helmert, 2006) ganhou a competição de planejamento de 2004, e o LAMA (Richter e Westphal, 2008), um planejador baseado em FASTDOWNWARD com heurísticas melhoradas, ganhou a competição de 2008.

Bylander (1994) e Ghallab *et al*. (2004) discutem a complexidade computacional de diversas variantes do problema de planejamento. Helmert (2003) prova os limites de complexidade para

muitos dos problemas *benchmark* e Hoffmann (2005) analisa o espaço de busca da heurística de ignorar as listas de exclusão. Heurísticas para o problema de cobertura de conjuntos para operações de agendamento da ferrovia italiana são discutidas por Caprara *et al.* (1995). Edelkamp (2009) e Haslum *et al.* (2007) descreveram como construir bases de dados-padrão para heurísticas de planejamento. Como mencionado no Capítulo 3, Felner *et al.* (2004) mostraram resultados encorajadores utilizando bases de dados-padrão para quebra-cabeças de blocos deslizantes, que pode ser tido como domínio de planejamento, mas Hoffmann *et al.* (2006) mostraram algumas limitações de abstração para os problemas de planejamento clássico.

Avrim Blum e Merrick Furst (1995, 1997) revitalizaram a área de planejamento com seu sistema GRAPHPLAN, ordens de grandeza mais rápido do que os planejadores de ordem parcial da época. Logo seguiram outros sistemas de grafo de planejamento, como o IPP (Koehler *et al.*, 1997), STAN (Fox e Long, 1998), e o SGP (Weld *et al.*, 1998). Um pouco antes, uma estrutura de dados se assemelhando ao grafo de planejamento foi desenvolvida por Ghallab e Laruelle (1994), usada pelo planejador de ordem parcial IXTET para derivar heurísticas precisas para orientar a busca. Nguyen *et al.* (2001) analisam cuidadosamente as heurísticas derivadas de grafos de planejamento. Nossa discussão sobre grafos de planejamento se baseia, em parte, nesse trabalho e em notas de aula e artigos de Subbarao Kambhampati (Bryce e Kambhampati, 2007). Como mencionado no capítulo, um grafo de planejamento pode ser usado de maneiras muito diferentes para orientar a busca por uma solução. O vencedor da competição de planejamento AIPS 2002, LGP (Gerevini e Serina, 2002, 2003), faz busca nos grafos de planejamento utilizando uma técnica de busca local inspirada no WALKSAT.

A abordagem do cálculo de situações para o planejamento foi introduzida por John McCarthy (1963). A versão mostrada aqui foi proposta por Ray Reiter (1991, 2001).

Kautz *et al.* (1996) pesquisaram várias maneiras de proposicionalizar esquemas de ação, verificando que as formas mais compactas nem sempre conduzem à solução mais rápida. Uma análise sistemática foi realizada por Ernst *et al.* (1997), que também desenvolveram um "compilador" automático para gerar representações proposicionais a partir de problemas PDDL. O planejador BLACKBOX, que combina ideias de GRAPHPLAN e SATPLAN, foi desenvolvido por Kautz e Selman (1998). O CPLAN, um planejador baseado em satisfação de restrições, foi descrito por Van Beek e Chen (1999).

Mais recentemente, surgiu o interesse na representação de planos como **diagrama de decisão binária**, estruturas de dados compactas de expressões booleanas, intensamente estudada na comunidade de verificação de hardware (Clarke e Grumberg, 1987; McMillan, 1993). Existem técnicas para demonstrar propriedades de diagramas de decisão binária, inclusive a propriedade de ser uma solução para um problema de planejamento. Cimatti *et al.* (1998) apresentam um planejador baseado nessa abordagem. Outras representações também foram usadas; por exemplo, Vossen *et al.* (2001) pesquisam o uso da programação inteira para planejamento.

A decisão final ainda não foi tomada, mas já existem algumas comparações interessantes entre as diversas abordagens para planejamento. Helmert (2001) analisa várias classes de problemas de planejamento e mostra que as abordagens baseadas em restrições, como GRAPHPLAN e SATPLAN, são melhores para domínios NP-difíceis, enquanto abordagens baseadas em busca funcionam melhor em domínios nos quais podem ser encontradas soluções possíveis sem retrocesso. O GRAPHPLAN e

o SATPLAN têm dificuldades em domínios com muitos objetos porque isso significa que eles têm de criar muitas ações. Em alguns casos, o problema pode ser retardado ou evitado gerando-se dinamicamente as ações proposicionalizadas, apenas quando necessário, em vez de instanciar todas elas antes do início da busca.

Readings in Planning (Allen *et al*., 1990) é uma antologia abrangente dos primeiros trabalhos nesse campo. Weld (1994, 1999) fornece duas excelentes revisões de algoritmos de planejamento dos anos 90. É interessante observar a mudança nos cinco anos entre as duas revisões: a primeira se concentra no planejamento de ordem parcial, enquanto a segunda introduz o GRAPHPLAN e o SATPLAN. *Automated Planning* (Ghallab *et al*., 2004) é um livro excelente sobre todos os aspectos do planejamento. O livro de LaValle, *Algoritmos de planejamento* (2006) abrange tanto o planejamento clássico como o estocástico, com ampla cobertura do planejamento do movimento de robô.

A pesquisa em planejamento foi central para a IA desde sua concepção, e os artigos sobre planejamento são frequentes nos principais periódicos e em conferências sobre IA. Também existem conferências especializadas, como a *International Conference on AI Planning Systems* (AIPS), o *International Workshop on Planning and Scheduling for Space*, e a *European Conference on Planning* (ECP). A partir de 2003, a *International Conference on Automated Planning and Scheduling* (ICAPS) uniu as duas conferências AIPS e ECP.

EXERCÍCIOS

10.1 Descreva as diferenças e as semelhanças entre a resolução de problemas e o planejamento.

10.2 Dados os esquemas de ação e o estado inicial da Figura 10.1, quais são todas as instâncias concretas aplicáveis de *Voar*(*p*, *de*, *para*) no estado descrito por:

$$Em(P_1, JFK) \land Em(P_2, SFO) \land Avião(P_1) \land Avião(P_2)$$
$$\land\ Aeroporto(JFK) \land Aeroporto(SFO)?$$

10.3 O problema do macaco e das bananas é enfrentado por um macaco em um laboratório com algumas bananas penduradas no teto, fora do alcance. Há uma caixa disponível que permitirá ao macaco alcançar as bananas se ele subir nela. Inicialmente, o macaco está em *A*, as bananas em *B* e a caixa em *C*. O macaco e a caixa têm a altura *Baixa*, mas, se subir na caixa, o macaco terá a altura *Alta*, a mesma altura das bananas. As ações disponíveis para o macaco incluem *Ir* de um lugar para outro, *Empurrar* um objeto de um lugar para outro, *Subir* em um objeto ou *Descer* de um objeto, e ainda *Pegar* ou *Soltar* um objeto. O resultado de *Agarrar* é que o macaco segura o objeto, se o macaco e o objeto estiverem no mesmo lugar e na mesma altura.

a. Escreva a descrição do estado inicial.

b. Escreva os seis esquemas de ação.

c. Suponha que o macaco queira enganar os cientistas que saíram para tomar chá, pegando as bananas, mas deixando a caixa em seu lugar original. Escreva isso como um objetivo geral (ou

seja, não considerando que a caixa esteja necessariamente em *C*) na linguagem do cálculo de situações. Esse objetivo pode ser resolvido por um sistema de planejamento clássico?

d. Seu esquema para empurrar provavelmente está incorreto porque, se o objeto for pesado demais, sua posição permanecerá a mesma quando o operador *Empurrar* for aplicado. Corrija seu esquema de ação para levar em conta objetos pesados.

10.4 O planejador STRIPS original foi projetado para controlar o robô Shakey. A Figura 10.14 mostra uma versão do mundo de Shakey que consiste em quatro salas dispostas ao longo de um corredor, onde cada sala tem uma porta e um interruptor de luz. As ações no mundo de Shakey incluem movimentar-se de um lugar para outro, empurrar objetos móveis (como caixas), subir e descer de objetos rígidos (como caixas) e ligar e desligar interruptores. O robô propriamente dito não poderia subir em uma caixa ou acionar um interruptor, mas o planejador STRIPS era capaz de descobrir e imprimir planos que estavam além das habilidades do robô. As seis ações de Shakey são as seguintes:

- *Ir*(x, y, r), que exige que Shakey esteja *em* x e que x e y sejam posições *na* mesma sala r. Por convenção, uma porta entre duas salas está em ambas as salas.
- Empurrar uma caixa b da posição x para a posição y dentro da mesma sala: *Empurrar*(b, x, y, r). Precisaremos do predicado *Caixa* e de constantes para as caixas.
- Subir em uma caixa de posição x: *Subir*(x, b); descer de uma caixa para a posição x: *Descer*(b, x). Precisaremos do predicado *Sobre* e da constante *Chão*.
- Ligar ou desligar um interruptor: *Ligar*(s, b) ; *Desligar*(s, b). Para ligar ou desligar um interruptor de luz, Shakey tem de estar em cima de uma caixa na posição do interruptor de luz.

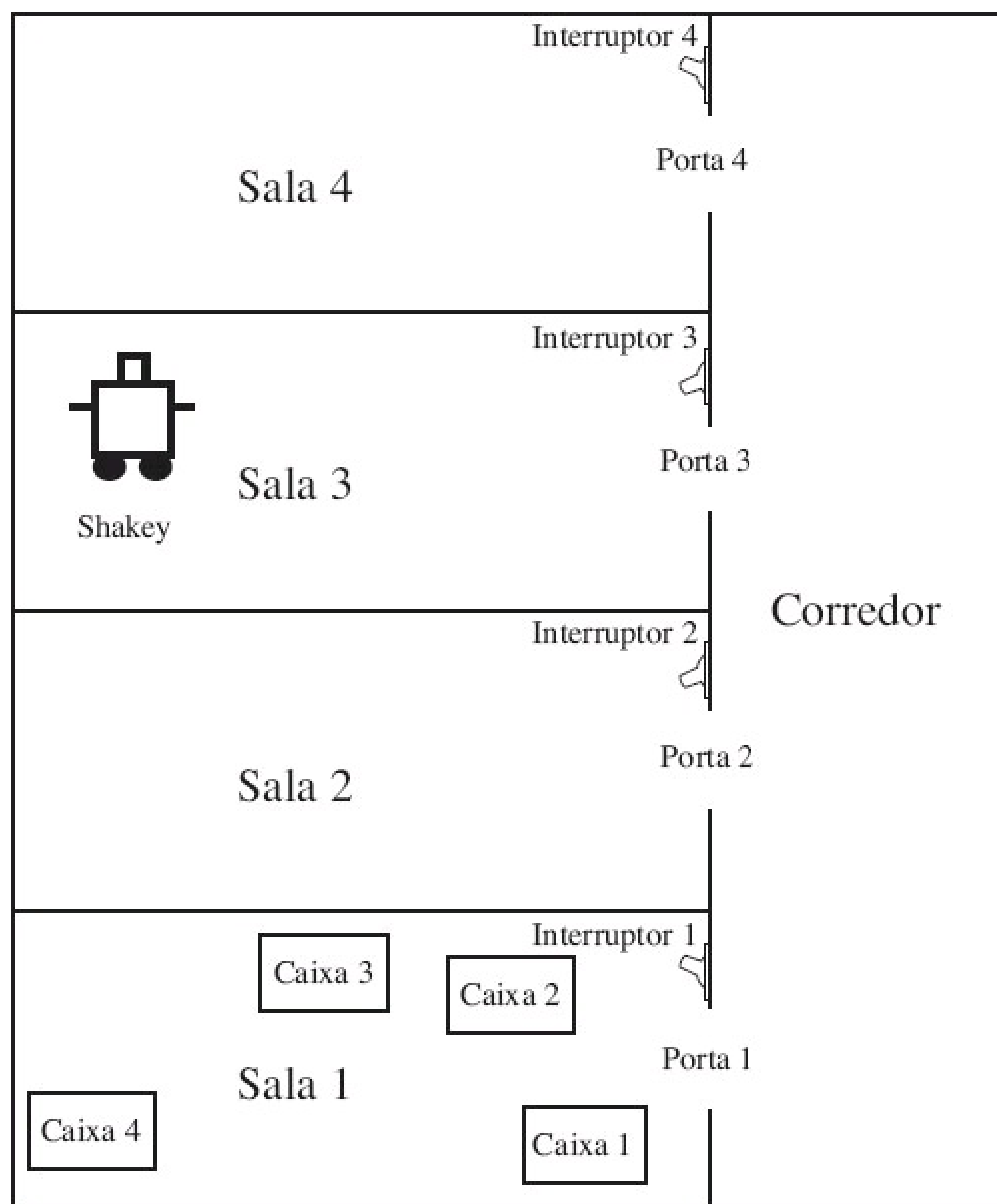


Figura 10.14 O mundo de Shakey. O robô Shakey pode se mover entre marcos dentro de uma sala, pode passar pela porta entre as salas, pode subir e descer de objetos que permitam essa ação e empurrar objetos que possam ser empurrados, e ainda pode ligar e desligar interruptores de luz.

Descreva PDDL sentenças para as seis ações de Shakey e o estado inicial da Figura 10.14. Construa um plano para Shakey colocar $Caixa_2$ na $Sala_2$.

10.5 Uma máquina de Turing finita tem uma fita de células finita unidimensional, cada célula que contém um símbolo de um número finito de símbolos. Uma célula tem uma cabeça de leitura e escrita sobre ela. Há um conjunto de estados finito onde a máquina pode estar, um dos quais é o estado de aceitação. Em cada passo, dependendo do símbolo na célula sob a cabeça e o estado atual da máquina, existe um conjunto de ações que podemos escolher. Cada ação envolve escrever um símbolo na célula sob a cabeça, fazendo a transição da máquina para um estado e opcionalmente o movimento da cabeça para a esquerda ou para a direita. O mapeamento que determina que ações são permitidas é o programa da máquina de Turing. O objetivo é o controle da máquina no estado de aceitação.

Represente o problema de aceitação da máquina de Turing como um problema de planejamento. Se isso pode ser feito, demonstre que determinar se um problema de planejamento tem uma solução é

pelo menos tão difícil como o problema de aceitação de Turing que é PSPACE-difícil.

10.6 Explique por que descartar efeitos negativos de todo esquema de ação em um problema de planejamento resulta em um problema relaxado.

10.7 A Figura 10.4 mostra um problema de mundo de blocos conhecido como **anomalia de Sussman**. O problema foi considerado anômalo porque os planejadores sem intercalações do início da década de 1970 não conseguiram resolvê-lo. Escreva uma definição do problema e resolva-o, manualmente ou com um programa de planejamento. Um planejador sem intercalações é um planejador que, ao receber dois subobjetivos G_1 e G_2, produz um plano para o G_1 concatenado com um plano para G_2 ou vice-versa. Explique por que um planejador sem intercalações não consegue resolver esse problema.

10.8 Prove que a busca para trás é completa com problemas PDDL.

10.9 Construa níveis 0, 1 e 2 do grafo de planejamento para o problema da Figura 10.1.

10.10 Prove as asserções a seguir sobre grafos de planejamento:

a. Um literal que não aparece no último nível do grafo não pode ser alcançado.

b. O custo de nível de um literal em um grafo serial não é maior que o custo real de um plano ótimo para alcançá-lo.

10.11 A heurística de nível de conjunto (consulte a Seção 10.3.1) utiliza um grafo de planejamento para estimar o custo de atingir o objetivo conjuntivo a partir do estado corrente. Para qual problema relaxado a heurística de nível de conjunto é uma solução?

10.12 Examine a definição de **busca bidirecional** no Capítulo 3.

a. A busca do espaço de estados bidirecional é uma boa ideia para o planejamento?

b. E sobre o emprego da busca no espaço de estados bidirecional para planejamento de ordem parcial?

c. Imagine uma versão de planejamento de ordem parcial no qual uma ação pode ser acrescentada a um plano se suas pré-condições puderem ser atingidas pelo efeito das ações que já estiverem no plano. Explique como lidar com conflitos e ordenação de restrições. O algoritmo é essencialmente idêntico à busca para frente em espaço de estados?

10.13 Comparamos planejadores de busca para a frente e para trás no espaço de estados a planejadores de ordem parcial, afirmando que um planejador de ordem parcial é um buscador no espaço de planos. Explique o quanto a busca para a frente e a busca para trás no espaço de estados também podem ser consideradas buscadores no espaço de planos e informe quais são os operadores de aprimoramento do plano.

10.14 Até aqui, supomos que os planos que criamos sempre se certificam de que as precondições de uma ação sejam satisfeitas. Vamos agora investigar o que os axiomas proposicionais de estado-sucessor tal como $TerSeta^{t+1} \Leftrightarrow (TerSeta^{t} \wedge \neg Atirar^{t})$ têm a dizer sobre ações cujas precondições não forem satisfeitas.

a. Mostre que os axiomas preveem que nada acontecerá quando uma ação for executada em um estado em que suas precondições não sejam satisfeitas.

b. Considere um plano *p* que contenha as ações exigidas para alcançar um objetivo, mas também inclua ações inválidas. É possível ocorrer o caso em que

estado inicial ∧ axiomas de estados sucessores ∧ p ⊨ objetivo ?

c. Com axiomas de estados sucessores de primeira ordem no cálculo de situações, é possível provar que um plano que contenha ações inválidas alcançará o objetivo?

10.15 Considere como converter um conjunto de esquemas de ações nos axiomas de estado-sucessor do cálculo de situações.

a. Considere o esquema para *Voar*(*p*, *de*, *para*). Escreva uma definição lógica para o predicado *Possível*(*Voar*(*p*, *de*, *para*) , *s*), que é verdadeiro se as precondições para *Voar*(*p*, *de*, *para*) forem satisfeitas na situação *s*.

b. Em seguida, supondo que *Voar*(*p*, *de*, *para*) seja o único esquema de ação disponível para o agente, escreva um axioma de estado-sucessor para *Em*(*p*, *x*, *s*) que capte as mesmas informações que o esquema de ação.

c. Agora, suponha que exista um método adicional de viagem: *Teleportar*(*p*, *de*, *para*). Ele tem a precondição adicional ¬*Danificado*(*p*) e o efeito adicional *Danificado*(*p*). Explique como a base de conhecimento do cálculo de situações deve ser modificada.

d. Por fim, desenvolva um procedimento geral e especificado com precisão para executar a conversão de um conjunto de esquemas STRIPS em um conjunto de axiomas de estado-sucessor.

10.16 No algoritmo SATPLAN da Figura 7.22, cada chamada ao algoritmo de satisfatibilidade confirma um objetivo g^T, onde T varia de 0 a T_{max}. Suponha que, em vez disso, o algoritmo de satisfatibilidade seja chamado apenas uma vez, com o objetivo $g^0 \lor g^1 \lor \ldots \lor g^{Tmax}$.

a. Isso sempre devolverá um plano se existir algum com tamanho menor ou igual a T_{max}?

b. Essa abordagem introduz alguma nova "solução" espúria?

c. Discuta como se poderia modificar um algoritmo de satisfatibilidade como WALKSAT, de forma que ele encontre soluções curtas (se elas existirem) ao receber um objetivo disjuntivo com essa forma.

---

[1] PDDL foi derivada da linguagem de planejamento original STRIPS (Fikes e Nilsson, 1971), ligeiramente mais restrita que a PDDL: as precondições e os objetivos do STRIPS não podem conter literais negativos.

[2] O mundo dos blocos utilizado em pesquisa de planejamento é muito mais simples do que a versão SHRDLU, mostrada no capítulo 1.

[3] Muitos problemas são escritos com essa convenção. Para os que não são, substituir todo o literal negativo ¬*P* em um objetivo ou precondição com um novo literal positivo *P*′.

CAPÍTULO 11

Planejamento e ação no mundo real

Em que vemos como representações mais expressivas e arquiteturas de agentes mais interativas resultam em planejadores que são úteis no mundo real.

O capítulo anterior introduziu os conceitos mais básicos, representações e algoritmos para planejamento. Os planejadores empregados no mundo real para planejar e escalonar as operações de espaçonaves, fábricas e campanhas militares são mais complexos; eles estendem a linguagem de representação e também o modo como o planejador interage com o ambiente. Este capítulo mostra como isso é feito. A Seção 11.1 estende a linguagem clássica para o planejamento para expressar ações com duração e restrição de recursos. A Seção 11.2 descreve métodos para a construção de planos organizados hierarquicamente. Isso permite que os especialistas humanos comuniquem ao planejador o que sabem sobre como resolver o problema. A hierarquia também se presta à construção eficiente do plano porque o planejador pode resolver um problema em nível abstrato, antes de se aprofundar em detalhes. A Seção 11.3 apresenta arquiteturas de agente que podem lidar com ambientes incertos e intercalar deliberação com execução, e dá alguns exemplos de sistemas do mundo real. A Seção 11.4 mostra como planejar quando o ambiente contém outros agentes.

11.1 TEMPO, ESCALONAMENTOS E RECURSOS

A representação do planejamento clássico informa *o que fazer* e *em que ordem*, mas a representação não pode informar sobre o tempo: *quanto tempo* uma ação leva e *quando* ela ocorre. Por exemplo, os planejadores do Capítulo 10 podem produzir um cronograma para uma companhia aérea que informa quais aviões estão designados a quais voos, mas também precisamos saber realmente os horários de partida e de chegada. Esse é o tema do **escalonamento**. O mundo real também impõe muitas **restrições de recursos**; por exemplo, uma companhia aérea tem número limitado de pessoal — e o pessoal designado para um voo não pode estar em outro ao mesmo tempo. Esta seção aborda os métodos para representar e resolver problemas de planejamento que incluem restrições temporais e de recursos.

A abordagem que adotamos nesta seção é “planejar primeiro, escalonar mais tarde”, isto é, dividimos o problema global em uma fase de *planejamento*, em que as ações são selecionadas com

algumas restrições de ordem para satisfazer aos objetivos do problema, e em uma fase de *escalonamento* posterior, na qual informações temporais são adicionadas ao plano, a fim de assegurar que ele atenderá às restrições de recursos e de prazos.

Processos({*AdicionarMotor1* $\prec$ *AdicionarRodas1* $\prec$ *Inspecionar1*},
{*AdcionarMotor2* $\prec$ *AdicionarRodas2* $\prec$ *Inspecionar2*})
Recursos(*GuinchoParaMotor*(1), *EstaçãodeRodas*(1), *Inspetores*(2), *PorcasDeRoda*(500))
Ação(*AdicionarMotor1*, DURAÇÃO:30,
USO: *GuinchoParaMotor*(1))
Ação(*AdicionarMotor2*, DURAÇÃO:60,
USO: *GuinchoParaMotor*(1))
Ação (*AdicionarMotor1*, DURAÇÃO:30,
CONSUMO: *PorcasDeRoda*(20), USO: *EstaçãodeRodas*(1))
Ação(*AdicionarMotor2*, DURAÇÃO:15,
CONSUMO: *PorcasDeRoda*(20), USO: *EstaçãodeRodas*(1))
Ação ($Inspecionar_i$, DURAÇÃO:10,
USO: *Inspetores*(1))

Figura 11.1 Um problema de escalonamento de linha de produção para montar dois carros, com restrições de recursos. A notação A $\prec$ B significa que a ação *A* deve preceder a ação *B*.

Essa abordagem é comum em configurações do mundo real de manufatura e logística, em que a fase de planejamento é realizada frequentemente por especialistas humanos. Os métodos automatizados do Capítulo 10 também podem ser utilizados para a fase de planejamento, desde que eles produzam planos apenas com o mínimo de ordenação de restrições necessárias para correção. O GRAPHPLAN (Seção 10.3), o SATPLAN (Seção 10.4.1) e os planejadores de ordem parcial (Seção 10.4.4) podem fazer isso; os métodos baseados em busca (Seção 10.2) produzem planos totalmente ordenados, mas podem ser facilmente convertidos em planos com restrições de ordenação minimais.

11.1.1 Representando restrições temporais e de recursos

Um **problema** típico de **escalonamento de linha de produção**, como introduzido pela primeira vez na Seção 6.1.2, consiste em um conjunto de processos (jobs), cada um dos quais é composto por um conjunto de **ações** com restrições de ordenação entre elas. Cada ação tem uma **duração** e um conjunto de restrições de recursos exigidos pela ação. Cada restrição especifica um *tipo* de recurso (por exemplo, parafusos, chaves de porcas ou pilotos), a quantidade necessária desse recurso e se esse recurso é **consumível** (por exemplo, os parafusos não estão mais disponíveis para uso) ou **reutilizável** (por exemplo, um piloto está ocupado durante um voo, mas ficará disponível quando o voo terminar). Os recursos também podem ser *produzidos* por ações com consumo negativo, incluindo ações de manufatura, crescimento e reabastecimento. Uma solução para um problema de escalonamento de linha de produção deve especificar os horários de início de cada ação e deve satisfazer todas as restrições de ordenação temporal e as restrições de recursos. Tal como acontece

com problemas de busca e planejamento, soluções podem ser avaliadas de acordo com uma função de custo, o que pode ser bastante complicado, com custos de recursos não lineares, custos dependentes do tempo de atraso, e assim por diante. Para simplificar, assumiremos que a função custo é a duração total do plano, que é chamado de **makespan.**

A Figura 11.1 mostra um exemplo simples: um problema que envolve a montagem de dois carros. O problema consiste em dois processos, cada um da forma [*AdicionarMotor*, *AdicionarRodas*, *Inspecionar*]. Em seguida, a instrução *Recursos* declara que existem quatro tipos de recursos e dá a quantidade de cada tipo disponível no início: 1 guincho para levantar o motor, 1 estação de rodas, 2 inspetores e 500 porcas de roda. Os esquemas de ação fornecem a duração e os recursos necessários para cada ação. As porcas de roda são *consumidas* à medida que as rodas são adicionadas ao carro, enquanto os outros recursos são "emprestados" no início de uma ação e liberados no final da ação.

A representação de recursos como quantidades numéricas, como *Inspetores*(2), em lugar de entidades nomeadas, como *Inspetor*(I_1) e *Inspetor*(I_2), é um exemplo de técnica muito geral chamada de **agregação**. A ideia central da agregação é agrupar objetos individuais em quantidade quando os objetos são todos indistinguíveis no que se refere ao propósito em questão. Em nosso problema de montagem, não importa *qual* inspetor inspeciona o carro, e portanto não há necessidade de fazer a distinção (a mesma ideia funciona no caso do problema dos missionários e canibais do Exercício 3.9). A agregação é essencial para reduzir a complexidade. Considere o que acontece quando é proposto um escalonamento que tem 10 ações *Inspecionar* concorrentes, mas há apenas nove inspetores disponíveis. Com os inspetores representados como quantidades, uma falha é detectada de imediato e o algoritmo realiza o retrocesso para tentar outro escalonamento. Com os inspetores representados como indivíduos, o algoritmo efetua o retrocesso para experimentar todas as 10 maneiras de atribuir inspetores para ações.

11.1.2 Solução de problemas de escalonamento

Começaremos considerando apenas o problema de escalonamento temporal, ignorando as restrições de recursos. Para minimizar o makespan (duração do plano), deveremos encontrar os tempos de início de todas as ações consistentes com as restrições de ordem fornecidas com o problema. É útil visualizar essas restrições de ordem como um grafo direcionado relativo às ações, como mostrado na Figura 11.2. Podemos aplicar o **método do caminho critico** (CPM — *critical path method*) para esse grafo, para determinar o tempo inicial e final possível de cada ação. Um **caminho** através de um grafo que representa um plano de ordem parcial é uma sequência de ações linearmente ordenadas começando em *Início* e terminado em *Término* (por exemplo, existem dois caminhos no plano de ordem parcial na Figura 11.2).

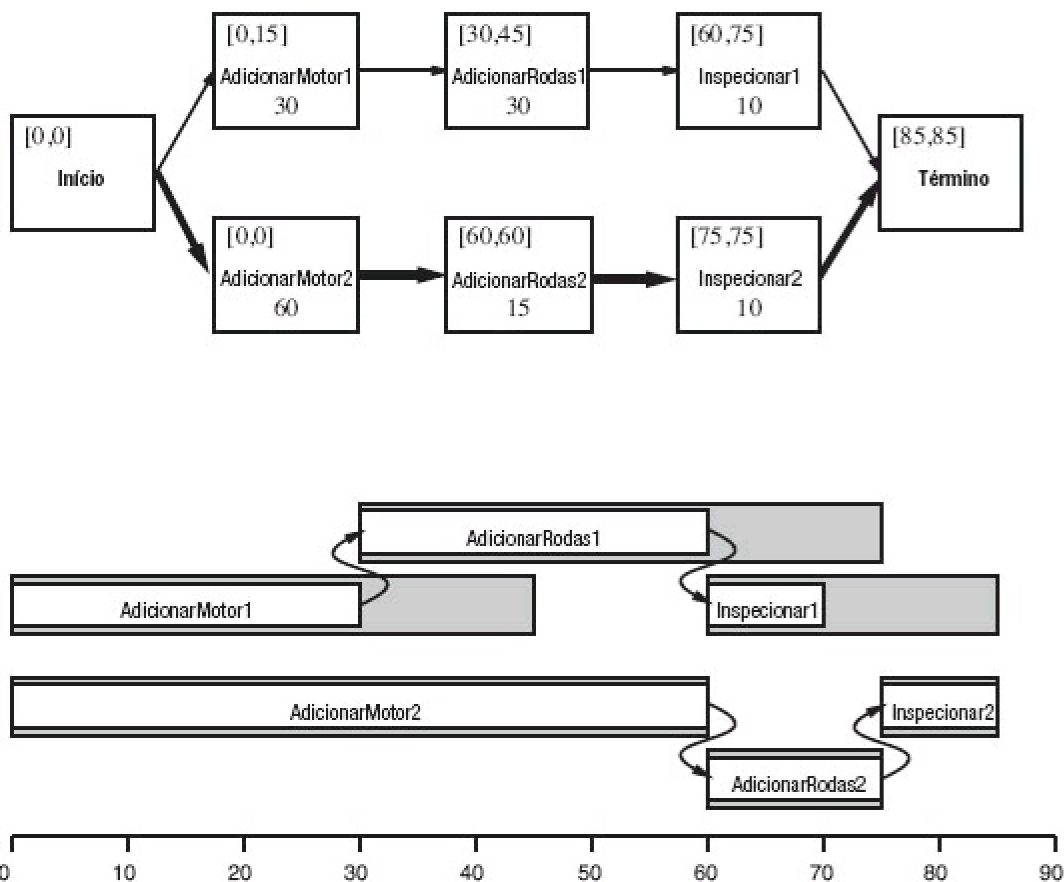


Figura 11.2 Parte superior: uma representação das restrições temporais para o escalonamento do problema da linha de produção Figura 11.1. A duração de cada ação é dada na parte inferior de cada retângulo. Na solução do problema, calculamos o tempos de início mais cedo e mais tardio como o par [*ES*, *LS*], exibido no canto superior esquerdo. A diferença entre esses dois números é a *folga de tempo* de uma ação; as ações com zero de folga estão no caminho crítico, como indica as setas em negrito. Parte inferior: a mesma solução mostrada na linha do tempo. Os retângulos cinza representam os intervalos de tempo durante os quais uma ação pode ser executada, desde que as restrições de ordem sejam respeitadas. A parte desocupada de um retângulo cinza indica uma folga de tempo.

O **caminho crítico** é aquele cuja duração total é a mais longa; o caminho é "crítico" porque determina a duração de todo o plano — encurtar outros caminhos não encurta o plano como um todo, mas o adiamento do início de qualquer ação no caminho crítico retarda o plano todo. As ações que estão fora do caminho crítico têm uma janela de tempo em que podem ser executadas. A janela é especificada em termos da hora de início mais cedo possível, *ES* (earliest start), e da mais tardia possível, *LS* (latest start). O intervalo entre *ES* e *LS* é conhecido como **folga** de uma ação. Podemos visualizar na Figura 11.2 que todo o plano vai levar 85 minutos, que cada ação no processo superior tem 15 minutos de folga e que cada ação no caminho crítico não tem folga (por definição). Juntos, o *ES* e o *LS* determinam o tempo para todas as ações, constituindo o **escalonamento** (plano com indicação de tempos e recursos) para o problema.

As fórmulas a seguir servem como uma definição para *ES* e *LS*, e também como o esboço de um algoritmo de programação dinâmica para calculá-los. *A* e *B* são ações, e $A \prec B$ significa que *A* vem antes de *B*:

$$
\begin{aligned}
&ES(Início) = 0 \\
&ES(B) = max_{A < B}\, ES(A) + Duração(A) \\
&LS(Término) = ES(Término) \\
&LS(A) = min_{B > A}\, LS(B) - Duração(A).
\end{aligned}
$$

A ideia é que comecemos atribuindo *ES*(Início) como 0. Então, logo que obtivermos a ação *B* tal que todas as ações que vêm imediatamente antes de *B* tenham valores *ES* atribuídos, estabeleceremos *ES*(*B*) como o máximo entre os tempos mais cedo de término das ações imediatamente anteriores, em que o tempo de término mais cedo de uma ação é definido como o tempo de início mais cedo, mais a duração. Esse processo é repetido até que tenha sido atribuído um valor *ES* a cada ação. Os valores *LS* são calculados de maneira semelhante, retrocedendo a partir da ação de *Término*.

A complexidade do algoritmo de caminho crítico é apenas $O(Nb)$, onde N é o número de ações e b é o fator máximo de ramificação entrando ou saindo de uma ação (para observar isso, note que os cálculos de *LS* e *ES* são efetuados uma vez para cada ação e cada cálculo itera, no máximo, b outras ações). Portanto, é bem fácil encontrar um escalonamento de mínima duração, dado um ordenamento parcial das ações e sem restrições de recursos.

Matematicamente falando, problemas de caminho crítico são fáceis de resolver porque são definidos como uma *conjunção* de inequações sobre os tempos de início e término. Quando introduzimos restrições de recursos, as restrições resultantes nos tempos de início e fim se tornam mais complicadas. Por exemplo, as ações *AdicionarMotor*, que começam ao mesmo tempo na Figura 11.2, exigem a mesma *GruaParaMotor* e, assim, não podem se sobrepor. A restrição “não podem se sobrepor” é uma *disjunção* de duas inequações lineares, uma para cada ordenação possível. A introdução de disjunções acaba por tornar o escalonamento com restrições de recursos NP-difícil.

A Figura 11.3 mostra a solução com o menor tempo de conclusão, 115 minutos, isto é, 30 minutos mais do que os 85 minutos necessários para um escalonamento sem restrições de recursos. Observe que nenhuma vez é necessário os dois inspetores; assim podemos mover um de nossos dois inspetores imediatamente para uma posição mais produtiva.

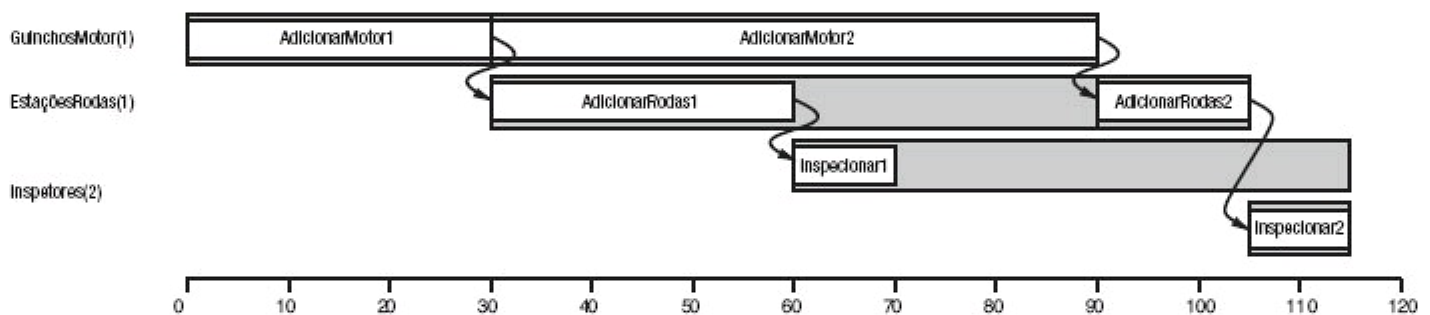


Figura 11.3 Uma solução para o problema de escalonamento de linha de produção da Figura 11.1, levando em conta a restrição de recursos. A margem à esquerda lista os três recursos reutilizáveis, e as ações são mostradas alinhadas horizontalmente com os recursos que elas utilizam. Há dois escalonamentos possíveis, dependendo de qual montagem utiliza o guincho para motor em primeiro lugar; mostramos a solução de menor duração, que leva 115 minutos.

A complexidade do escalonamento com restrições de recurso é sempre vista, tanto na prática quando na teoria. Um problema desafiante proposto em 1963 — encontrar o escalonamento ótimo

para um problema envolvendo apenas 10 máquinas e 10 processos de 100 ações cada — permaneceu sem solução por 23 anos (Lawler *et al*., 1993). Muitas abordagens foram experimentadas, inclusive ramificar-e-limitar, têmpera simulada, busca tabu, satisfação de restrições e outras técnicas dos Capítulos 3 e 4. Uma heurística simples, mas popular, é o algoritmo de **folga mínima:** em cada iteração, o escalonamento com o tempo mais cedo possível, é iniciado para qualquer ação não escalonada que tenha todos os seus predecessores escalonados, e com a mínima folga; então, atualiza-se os horários de *ES* e *LS* para cada ação afetada e, em seguida, repete-se a tarefa. A heurística se assemelha à heurística de mínimos valores restantes (MVR) heurísticos na satisfação de restrição. Com frequência, ela funciona bem na prática, mas, em nosso problema de montagem, produz uma solução de 130 minutos, e não a solução de 115 minutos da Figura 11.3.

Até este ponto, assumimos que o conjunto de ações e de restrições de ordenação é fixo. Sob esse pressuposto, todos os problemas de escalonamento podem ser resolvidos por uma sequência sem sobreposição que evita todos os conflitos de recursos, desde que cada ação seja viável por si só. Se um problema de escalonamento mostrar-se muito difícil, no entanto, pode não ser uma boa ideia resolvê-lo dessa forma — pode ser melhor reconsiderar as ações e as restrições, caso conduza a um problema de escalonamento muito mais fácil. Nesse caso, faz sentido *integrar* planejamento e escalonamento, levando em conta durações e sobreposições durante a construção de um plano de ordem parcial. Vários algoritmos de planejamento do Capítulo 10 podem ser estendidos para lidar com essas informações. Por exemplo, os planejadores de ordem parcial podem detectar violações de restrições de recursos de modo quase idêntico à forma como detectam conflitos com vínculos causais. As heurísticas podem ser projetadas para avaliar o tempo de conclusão total de um plano. Essa é atualmente uma área ativa de busca.

11.2 PLANEJAMENTO HIERÁRQUICO

Os métodos de resolução de problemas e de planejamento dos capítulos anteriores operam todos com um conjunto fixo de ações atômicas. As ações podem ser agrupadas em sequências ou em redes ramificadas; os algoritmos de ponta podem gerar soluções contendo milhares de ações.

Para os planos executados pelo cérebro humano, as ações atômicas são ativações musculares. Fazendo um arredondamento, temos cerca de 10^3 músculos para ativar (alguns contam 639, mas muitos deles têm múltiplas subunidades); podemos modular a sua ativação, talvez 10 vezes por segundo; e estamos vivos e despertos por cerca de 10^9 segundos ao todo. Assim, uma vida humana contém cerca de 10^{13} ações, tirando ou adicionando uma ou duas ordens de magnitude. Mesmo se nos limitarmos a planejar para horizontes muito mais curtos de tempo, por exemplo, férias de duas semanas no Havaí, um plano de movimentação detalhado conterá cerca de 10^{10} ações. Isso é muito mais do que 1.000.

Para preencher essas lacunas, os sistemas de IA provavelmente terão que fazer o que os seres humanos parecem fazer: planejar em níveis mais altos de abstração. Um plano razoável para as férias no Havaí poderia ser “Ir ao aeroporto de San Francisco; tomar o voo 11 da Hawaiian Airlines para Honolulu, fazer o que se faz nas férias durante duas semanas, pegar o voo 12 da Hawaiian Airlines de volta para San Francisco; ir para casa”. Dado um plano desse tipo, a ação “Vá para o aeroporto

de San Francisco” pode ser vista como uma tarefa de planejamento em si, com uma solução, como “Dirija para o estacionamento com pernoite; estacione; pegue o transfer (serviço de transporte do aeroporto) para o terminal”. Cada uma dessas ações, por sua vez, pode ser ainda decomposta até chegarmos ao nível de ações que podem ser executadas sem deliberação para gerar as sequências motoras de controle exigidas.

Nesse exemplo, vemos que o planejamento pode ocorrer tanto antes quanto durante a execução do plano; por exemplo, provavelmente, poder-se-ia adiar o problema do planejamento da rota da vaga no estacionamento com pernoite para o serviço de transfer até que seja encontrada uma vaga de estacionamento durante a execução. Assim, essa ação em particular permanecerá em um nível abstrato antes da fase de execução. Adiaremos a discussão desse tópico até a Seção 11.3. Aqui nos concentraremos no aspecto de **decomposição hierárquica**, uma ideia que permeia quase todas as tentativas de gerenciar a complexidade. Por exemplo, a criação de um software complexo pode ser feita a partir de uma hierarquia de sub-rotinas ou classes de objetos; exércitos operam como uma hierarquia de unidades; governos e corporações têm hierarquias de departamentos, subsidiárias e filiais. O principal benefício da estrutura hierárquica é que, em cada nível da hierarquia, uma tarefa computacional, missão militar ou função administrativa é reduzida a um *pequeno* número de atividades, no próximo nível abaixo, de modo que o custo computacional de encontrar a maneira correta de organizar as atividades para o problema atual é baixo. Os métodos não hierárquicos, por outro lado, reduzem uma tarefa a um número *grande* de ações individuais e, para problemas de grande escala, isso fica completamente impraticável.

11.2.1 Ações de alto nível

O formalismo básico que adotamos para entender a decomposição hierárquica vem da área das **redes hierárquicas de tarefa** ou de planejamento HTN. Como no planejamento clássico (Capítulo 10), assumimos observabilidade e determinismo total e a disponibilidade de um conjunto de ações, agora chamadas de **ações primitivas**, com esquemas padrão de precondições-efeito. O conceito-chave adicional é a **ação de alto nível** ou HLA (high-level action) — por exemplo, a ação “Vá ao aeroporto de San Francisco”, no exemplo dado anteriormente. Cada HLA tem um ou mais **refinamentos** possíveis, em uma sequência[1] de ações, cada uma das quais pode ser uma HLA ou uma ação primitiva (que não tem refinamentos por definição). Por exemplo, a ação “Ir ao aeroporto de San Francisco”, representada formalmente como *Ir*(*Casa, SFO*), pode ter dois refinamentos possíveis, como mostrado na Figura 11.4. A mesma figura mostra um refinamento **recursivo** para a navegação no mundo do aspirador de pó: para chegar a um destino, dar um passo e depois ir para o destino.

Refinamento(*Ir* (*Casa, SFO*),
  PASSOS: [*Dirigir*(*Casa, SFOEstacionamentoComPernoite*),
        Transfer(*SFOEstacionamentoComPernoite, SFO*)])
Refinamento(*Ir*(*Casa, SFO*),
  PASSOS: [*Taxi*(*Casa, SFO*)])

Refinamento(*Navegar* ([*a, b*], [*x, y*]),
PRECOND: $a = x \land b = y$
PASSOS: [])
Refinamento(*Navegar*([*a, b*], [*x, y*]),
PRECOND: *Conectado*([*a, b*], [*a* − *1, b*])
PASSOS: [*Esquerda, Navegar* ([*a* − *1, b*], [*x, y*])])
Refinamento(*Navegar*([*a, b*], [*x, y*]),
PRECOND: *Conectado*([*a, b*], [*a* + *1, b*])
PASSOS: [*Direito, Navegar*([*a* + *1, b*], [*x, y*])])
. . .

Figura 11.4 Definições de refinamentos possíveis de duas ações de alto nível: ir para o aeroporto de San Francisco e navegar no mundo do aspirador de pó. Neste último caso, observe a natureza recursiva do refinamento e a utilização das precondições.

Esses exemplos mostram que as ações de alto nível e seus refinamentos incorporam conhecimentos sobre *como fazer as coisas*. Por exemplo, os refinamentos *Ir*(*Casa, SFO*) diz que, para você chegar ao aeroporto, pode dirigir ou tomar um táxi; comprar leite, as ações sentar-se e mover o cavalo para e4 não devem ser considerados.

Um refinamento HLA que contém apenas ações primitivas é chamado de **implementação** de HLA. Por exemplo, no mundo do aspirador de pó, ambas as sequências [*Direita, Direita, Baixo*] e [*Baixo, Direita, Direita*] implementam a HLA *Navegar*([1, 3], [3 2]). Uma implementação de um plano de alto nível (uma sequência de HLAs) é a concatenação de implementações de cada HLA na sequência. Dadas as definições de precondição-efeito de cada ação primitiva, é simples determinar se qualquer implementação de um plano de alto nível atingiu o objetivo. Podemos dizer, então, *que um plano de alto nível atingiu o objetivo a partir de um dado estado se pelo menos uma de suas implementações atingiu o objetivo daquele estado.* "*Pelo menos uma*" é crucial nessa definição — nem *todas* as implementações precisam alcançar o objetivo porque o agente tem que decidir qual implementação vai executar. Assim, o conjunto de implementações possíveis no planejamento de HTN — cada um dos quais pode ter um resultado diferente — não é o mesmo que o conjunto de resultados possíveis no planejamento não determinístico. Lá, é necessário que um plano funcione para *todos* os resultados e, como o agente não pode escolher o resultado, a natureza o faz.

O caso mais simples é uma HLA que tenha exatamente uma implementação. Nesse caso, podemos computar as precondições e os efeitos da HLA a partir da implementação (veja o Exercício 11.3) e, em seguida, tratar a HLA exatamente como se fosse uma ação primitiva por si só. Pode ser constatado que a coleção correta de HLAs pode resultar na complexidade de tempo da busca cega caindo de exponencial para linear em profundidade da solução, embora a concepção da referida coleção de HLAs possa não ser uma tarefa trivial em si. Quando as HLAs têm várias implementações possíveis, existem duas opções: uma é buscar entre as implementações uma que funcione, como na Seção 11.2.2; outra é inferir diretamente sobre a HLA — apesar da multiplicidade de implementações —, como explicado na Seção 11.2.3. O último método permite a derivação de

planos abstratos comprovadamente corretos, sem a necessidade de considerar as suas implementações.

11.2.2 Busca por soluções primitivas

O planejamento de HTN muitas vezes é formulado com uma única ação de "nível superior" chamada *Ação*, em que o objetivo é encontrar uma implementação da *Ação* que alcance o objetivo. Essa abordagem é inteiramente genérica. Por exemplo, problemas de planejamento clássico podem ser definidos da seguinte forma: para cada ação primitiva a_i, proporcionar um refinamento da *Ação* com os passos [a_i, *Ação*]. Isso cria uma definição recursiva de *Ação* que nos permite adicionar ações. Mas precisamos de alguma forma de parar a recursão; fazemos isso fornecendo mais um refinamento da *Ação*, com uma lista vazia de passos e com uma precondição igual ao objetivo do problema. Isso significa que, se o objetivo já tiver sido alcançado, então a implementação correta é não fazer nada.

A abordagem leva a um algoritmo simples: escolher repetidamente uma HLA no plano atual e substituí-la por um de seus refinamentos, até que o plano atinja o objetivo. Uma implementação possível com base em busca em largura em árvore é mostrada na Figura 11.5. Planos são considerados em ordem de profundidade da rede de refinamentos, em vez do número de passos primitivos. É simples projetar uma versão do algoritmo de busca em grafo, bem como da busca em profundidade e das versões de aprofundamento iterativos.

```
função BUSCA-HIERÁRQUICA (problema, hierarquia) retorna uma solução ou falha
  fronteira ← uma fila FIFO com [Ação] como elemento único
  fim de laço
    se VAZIO? (fronteira), então retornar falha
    plano ← POP(fronteira) /*escolhe o plano menos profundo na fronteira*/
    HLA ← a primeira HLA no plano ou nulo se nenhuma
    prefixo,sufixo ← as subsequências de ações antes e depois da HLA no plano
    consequência ← RESULTADO(problema.ESTADO-INITIAL, prefixo)
    se HLA for nula então, / * então o plano é primitivo e consequência e o resultado * /
      se consequência satisfaz problema.OBJETIVO então retornar plano
    senão para cada sequência em REFINAMENTOS(HLA, consequência, hierarquia) faça
      fronteira ← INSIRA(CONCATENA(prefixo, sequência, sufixo), fronteira)
```

Figura 11.5 Uma implementação de busca em largura de planejamento hierárquico para a frente. O plano inicial fornecido ao algoritmo é [*Ação*]. A função REFINAMENTOS retorna um conjunto de sequências de ação, uma para cada refinamento da HLA cujas precondições são satisfeitas pelo estado especificado, *consequência*.

Em essência, essa forma de busca hierárquica explora o espaço de sequências que está de acordo com o conhecimento contido na biblioteca da HLA sobre como as coisas devem ser feitas. Pode ser

codificada uma grande parte do conhecimento, não apenas nas sequências de ação especificadas em cada refinamento, mas também nas precondições dos refinamentos. Para alguns domínios, os planejadores de HTN foram capazes de gerar grandes planos com muito pouca busca. Por exemplo, o O-PLAN (Bell e Tate, 1985), que combina planejamento HTN com escalonamento, foi utilizado para desenvolver planos de produção para a Hitachi. Um problema típico envolve uma linha de 350 produtos diferentes, 35 máquinas de montagem e mais de 2.000 operações diferentes. O planejador gera um escalonamento de 30 dias com três turnos de oito horas por dia, envolvendo dezenas de milhões de passos. Outro aspecto importante dos planos de HTN é que eles são, por definição, hierarquicamente estruturados; geralmente isso torna fácil para os seres humanos entenderem.

Os benefícios computacionais da busca hierárquica podem ser notados examinando um caso idealizado. Suponha que um problema de planejamento tenha uma solução com d ações primitivas. Para um planejador não hierárquico, de espaço de estados para a frente com b ações permitidas em cada estado, o custo será $O(b^d)$, conforme explicado no Capítulo 3. Para um planejador HTN, vamos supor uma estrutura de refinamento bem regular: cada ação não primitiva tem r refinamentos possíveis, cada um decompõe a ação em k ações no nível imediatamente inferior. Queremos saber quantas árvores de refinamento diferente existem com essa estrutura. Agora, se há d ações no nível primitivo, o número de níveis abaixo da raiz é $\log_k d$, de modo que o número de nós de refinamento internos é $1 + k + k^2 + \cdots + k^{\log_k d-1} = (d-1)/(k-1)$. Cada nó interno tem r refinamentos possíveis, então poderiam ser construídos $r^{(d-1)/(k-1)}$ árvores de decomposição regulares possíveis. Examinando essa fórmula, vemos que manter r pequeno e k grande pode resultar em grandes economias: essencialmente estaremos extraindo a raiz k-ésima do custo não hierárquico, e b e r são comparáveis. O r pequeno e o k grande significam uma biblioteca de HLAs com um pequeno número de refinamentos cada um produzindo uma longa sequência de ações (que, todavia, nos permite resolver qualquer problema). Isso nem sempre é possível: longas sequências de ações que são utilizáveis em ampla gama de problemas são extremamente preciosas.

A chave para o planejamento HTN, então, é a construção de uma biblioteca de planos contendo métodos conhecidos de implementação complexa, ações de alto nível. Um método de construção dessa biblioteca é *aprender* os métodos a partir da experiência de resolução de problemas. Depois da experiência dolorosa da construção de um plano a partir do zero, o agente pode salvar o plano na biblioteca como um método de implementação de ação de alto nível definido pela tarefa. Dessa forma, o agente pode se tornar cada vez mais competente ao longo do tempo à medida que sejam construídos novos métodos para substituir os métodos antigos.

Um aspecto importante desse processo de aprendizagem é a capacidade de *generalizar* os métodos que são construídos, eliminando detalhes específicos para a instância do problema (por exemplo, o nome do construtor ou o endereço do terreno) mantendo apenas os elementos principais do plano. No Capítulo 19 há uma descrição dos métodos para atingir esse tipo de generalização. Parece-nos inconcebível que os seres humanos possam ser tão competentes como são sem alguns desses mecanismos.

11.2.3 Busca por soluções abstratas

O algoritmo de busca hierárquica da seção anterior refina as HLAs por todo o caminho até as sequências de ações primitivas para determinar se um plano é viável. Isso contradiz o senso comum: ele deveria ser capaz de determinar que o plano de alto nível com duas HLAs:

[*Dirigir*(*Casa, SFOEstacionamentoCom Pernoite*),
Transfer(*SFOEstacionamentoComPernoite, SFO*)]

chega ao aeroporto sem ter que determinar uma rota precisa, para a escolha de vaga de estacionamento, e assim por diante. A solução parece óbvia: escrever descrições de precondição-efeito das HLAs, da mesma forma que escrevemos o que as ações primitivas fazem. Das descrições, deveria ser fácil provar que o plano de alto nível atingiu o objetivo. Esse é o Santo Graal, por assim dizer, do planejamento hierárquico porque, se derivamos um plano de alto nível que comprovadamente atinge o objetivo, trabalhando em um pequeno espaço de busca de ações de alto nível, então podemos nos comprometer com esse plano e trabalhar com o problema do refinamento de cada passo do plano. Isso nos dá a redução exponencial que buscamos. Para que isso funcione, todo o plano de alto nível que "reivindica" atingir o objetivo (em virtude das descrições de seus passos) de fato deve atingir o objetivo no sentido definido anteriormente: deve ter pelo menos uma implementação que atinja o objetivo. Essa propriedade chama-se **propriedade de refinamento descendente** para descrições HLA.

Escrever as descrições de HLA que satisfaçam a propriedade de refinamento descendente é simples, em princípio: uma vez que as descrições são *verdadeiras*, qualquer plano de alto nível que pretenda alcançar o objetivo deve de fato fazer isso — caso contrário, as descrições estão fazendo alguma reivindicação falsa sobre o que as HLAs fazem. Já vimos como escrever descrições verdadeiras para as HLAs que tenham exatamente uma implementação (Exercício 11.3), surge um problema quando a HLA tem múltiplas implementações. Como podemos descrever os efeitos de uma ação que pode ser implementada de muitas maneiras diferentes?

Uma resposta segura (pelo menos para os problemas em que todas as precondições e objetivos são positivos) é incluir apenas os efeitos positivos que são alcançados por *cada* implementação da HLA e os efeitos negativos de *qualquer* implementação. Em seguida, a propriedade de refinamento descendente seria satisfeita. Infelizmente, essa semântica para HLAs é muito conservadora. Considere novamente a *HLA Ir*(*Casa, SFO*), que tem dois refinamentos, e suponha, para simplificar a explicação, um mundo simples em que se pode sempre dirigir até o aeroporto e estacionar, mas tomar um táxi requer *Dinheiro* como precondição. Nesse caso, *Ir*(*Casa, SFO*) nem sempre o leva para o aeroporto. Em particular, ocorre falha se *Dinheiro* for falso, e por isso não podemos afirmar *Em*(*Agente, SFO*) como um efeito da HLA. Isso não faz sentido; no entanto, se o agente não tivesse *Dinheiro*, ele mesmo dirigiria. Exigir que um efeito seja válido para *cada* implementação é equivalente a assumir que *alguém*, um adversário, vá escolher a implementação. Isso trata as múltiplas consequências da HLA exatamente como se a HLA fosse **não determinística**, como na Seção 4.3. Para o nosso caso, o próprio agente vai escolher a implementação.

As comunidades de linguagens de programação cunharam a expressão **não determinismo demoníaco** para o caso em que um adversário faz as escolhas, contrastando com o **não determinismo angélico,** onde o próprio agente faz as escolhas. Tomamos emprestada essa expressão para definir a semântica angélica das descrições da HLA. O conceito básico necessário para a

compreensão da semântica angélica é o **conjunto alcançável** de uma HLA: dado um estado *s*, o conjunto alcançável de uma HLA *h*, escrita como ALCANÇAR(*s, h*), é o conjunto de estados alcançáveis por qualquer uma das implementações da HLA. A ideia fundamental é que o agente possa escolher em *qual* elemento do conjunto alcançável ele acaba quando executa a HLA; assim, uma HLA com refinamentos múltiplos é mais "poderosa" que a mesma HLA com menos refinamentos. Podemos também definir o conjunto alcançável de uma sequência de HLAs. Por exemplo, o conjunto alcançável de uma sequência $[h_1, h_2]$ é a união de todos os conjuntos alcançáveis obtidos através da aplicação de h_2 em cada estado no conjunto alcançável de h_1:

$$\text{ALCANCE}(s, [h_1, h_2]) = \bigcup_{s' \in \text{ALCANCE}(s, h_1)} \text{ALCANCE}(s' h_2).$$

Dadas essas definições, um plano de alto nível — uma sequência de HLAs — atinge o objetivo se o seu conjunto alcançável tem interseção o conjunto de estados objetivos (compare isso com uma condição muito mais forte para a semântica demoníaca, em que cada membro do conjunto alcançável tem de ser um estado objetivo). Por outro lado, se o conjunto alcançável não tem interseção com o objetivo, definitivamente o plano não funcionará. A Figura 11.6 ilustra essas ideias.

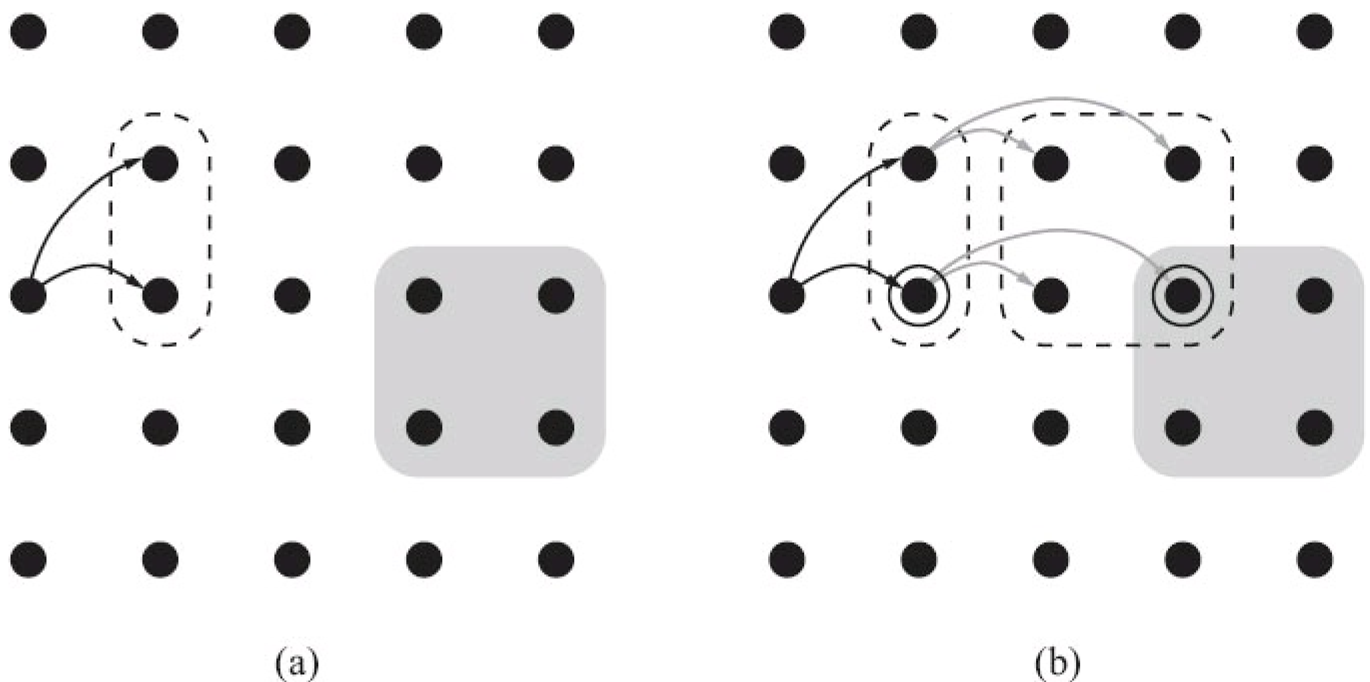


Figura 11.6 Exemplos esquemáticos de conjuntos alcançáveis. O conjunto de estados objetivo está sombreado. As flechas pretas e cinzas indicam possíveis implementações de h1 e h2, respectivamente. (a) O conjunto alcançável de um HLA h1 em um estado s. (b) O conjunto alcançável para a sequência [h1, h2]. Como ele tem interseção com o conjunto objetivo, a sequência alcança o objetivo.

A noção de conjuntos alcançáveis produz um algoritmo simples: busca entre planos de alto nível, procurando aquele cujo conjunto alcançável tem interseção com o objetivo; uma vez que isso acontece, o algoritmo pode *comprometer-se* com esse plano abstrato, sabendo que ele funciona, e se dedicar em refinar mais o plano. Vamos voltar para as questões algorítmicas mais tarde; em primeiro lugar, consideremos a questão de como os efeitos de uma HLA — o conjunto alcançável para cada

estado inicial possível — são representados. Tal como acontece com os esquemas de ação clássicos do Capítulo 10, representaremos as *mudanças* feitas em cada fluente. Pense em um fluente como uma variável de estado. A ação primitiva pode *adicionar* ou *excluir* uma variável ou deixá-la *inalterada* (com efeitos condicionais — consulte a Seção 11.3.1 —, há uma quarta possibilidade: lançar uma variável para o seu oposto).

Uma HLA sob semântica angélica pode fazer mais: pode *controlar* o valor de uma variável, definindo-a como verdadeira ou falsa, dependendo de que implementação for escolhida. Na verdade, uma HLA pode ter nove efeitos diferentes sobre uma variável: se a variável iniciar em verdadeiro, pode mantê-la sempre como verdadeira, sempre torná-la falso ou ter uma escolha; se a variável iniciar como falso, pode manter-se sempre como falso, sempre torná-la verdadeiro ou ter uma escolha; e as três opções de cada caso podem ser combinadas arbitrariamente, ficando nove no total. Como notação, isso é um pouco desafiador. Usaremos o símbolo ~ para significar "possivelmente, se o agente assim o desejar". Assim, um efeito $\tilde{+}A$ significa "possivelmente adicionar *A*", isto é, deixe *A* inalterado ou torne-o verdadeiro. Da mesma forma, $\tilde{-}A$ significa "possivelmente excluir *A*" e $\tilde{\pm}A$ significa "possivelmente adicionar ou excluir A". Por exemplo, a HLA *Ir*(*Casa, SFO*), com os dois refinamentos mostrados na Figura 11.4, possivelmente exclui *Dinheiro* (se o agente decidir tomar um táxi), por isso deveria ter o efeito $\tilde{-}$*Dinheiro*. Assim, verificamos que as descrições das HLAs são *deriváveis*, em princípio, das descrições de seus refinamentos — na verdade, isso é necessário se quisermos descrições de HLA verdadeiras, de tal forma que a propriedade de refinamento descendente seja válida. Agora, suponha que tenhamos os esquemas a seguir para as HLAs h_1 e h_2:

$$Ação(h_1, \text{PRECOND: } \neg A, \text{EFFECT } A \wedge \tilde{-}B),$$
$$Ação(h_2, \text{PRECOND: } \neg B, \text{EFFECT: } \tilde{+}A \wedge \tilde{\pm}C).$$

Ou seja, h_1 adiciona *A* e é possível que exclua *B*, enquanto h_2 possivelmente adiciona *A* e tem total controle sobre *C*. Agora, se apenas *B* for verdadeiro no estado inicial e o objetivo for $A \wedge C$, então a sequência $[h_1, h_2]$ alcança o objetivo: escolhemos uma implementação de h_1 que torna *B* falso, em seguida escolhemos uma implementação de h_2 que deixa *A* verdadeiro e torna C verdadeiro.

A discussão anterior assume que os efeitos de uma HLA — o conjunto alcançável para qualquer estado inicial dado — pode ser descrito exatamente descrevendo o efeito de cada variável. Seria bom se isso fosse sempre verdadeiro, mas em muitos casos só podemos aproximar os efeitos porque uma HLA pode ter infinitas implementações e pode produzir conjuntos alcançáveis arbitrariamente sinuosos — um pouco como o problema de estado de crença sinuoso ilustrado na Figura 7.21. Por exemplo, dissemos que *Ir*(*Casa, SFO*) possivelmente exclui *Dinheiro*; mas também adiciona possivelmente *Em*(*Carro, SFOEstacionamentoComPernoite*); mas não pode fazer as duas coisas — de fato, deve fazer exatamente uma. Tal como acontece com os estados de crença, talvez seja necessário escrever *descrições aproximadas*. Utilizaremos dois tipos de aproximação: uma **descrição otimista** ALCANCE$^+$(s, h) de uma HLA h pode exagerar o conjunto alcançável, enquanto uma **descrição pessimista** ALCANCE$^-$(s, h) pode subestimar o conjunto alcançável. Assim, temos

$$\text{ALCANCE}^-(s, h) \subseteq \text{ALCANCE}(s, h) \subseteq \text{ALCANCE}^+(s, h).$$

Por exemplo, uma descrição otimista de *Ir*(*Casa, SFO*) informa que é possível excluir *Dinheiro e*

possivelmente acrescentar *Em*(*Carro, SFOEstacionamentoComPernoite*). Outro bom exemplo surge do quebra-cabeça de oito peças, do qual metade dos estados é inalcançável de qualquer estado dado (consulte o Exercício 3.4): a descrição otimista de *Ação* poderia muito bem incluir o espaço de estados inteiro, uma vez que o conjunto exato alcançável é bastante sinuoso.

Com descrições aproximadas, o teste para saber se um plano alcança o objetivo precisa ser um pouco modificado. Se o conjunto alcançável otimista do plano não tem interseção com o objetivo, o plano não funciona; se o conjunto alcançável pessimista tem interseção com o objetivo, o plano funciona (Figura 11.7(a)). Com descrições exatas, um plano pode ou não funcionar, mas, com descrições aproximadas, há um meio-termo: se o conjunto otimista tem interseção com o objetivo, mas o pessimista não, não podemos dizer se o plano funciona (Figura 11.7(b)). Quando surge essa circunstância, a incerteza pode ser resolvida através do refinamento do plano. É uma situação muito comum no raciocínio humano. Por exemplo, no planejamento das duas semanas de férias no Havaí, anteriormente referido, alguém poderia propor passar dois dias em cada uma das sete ilhas. A prudência indicaria que esse plano ambicioso precisa ser refinado acrescentando detalhes de transporte entre as ilhas.

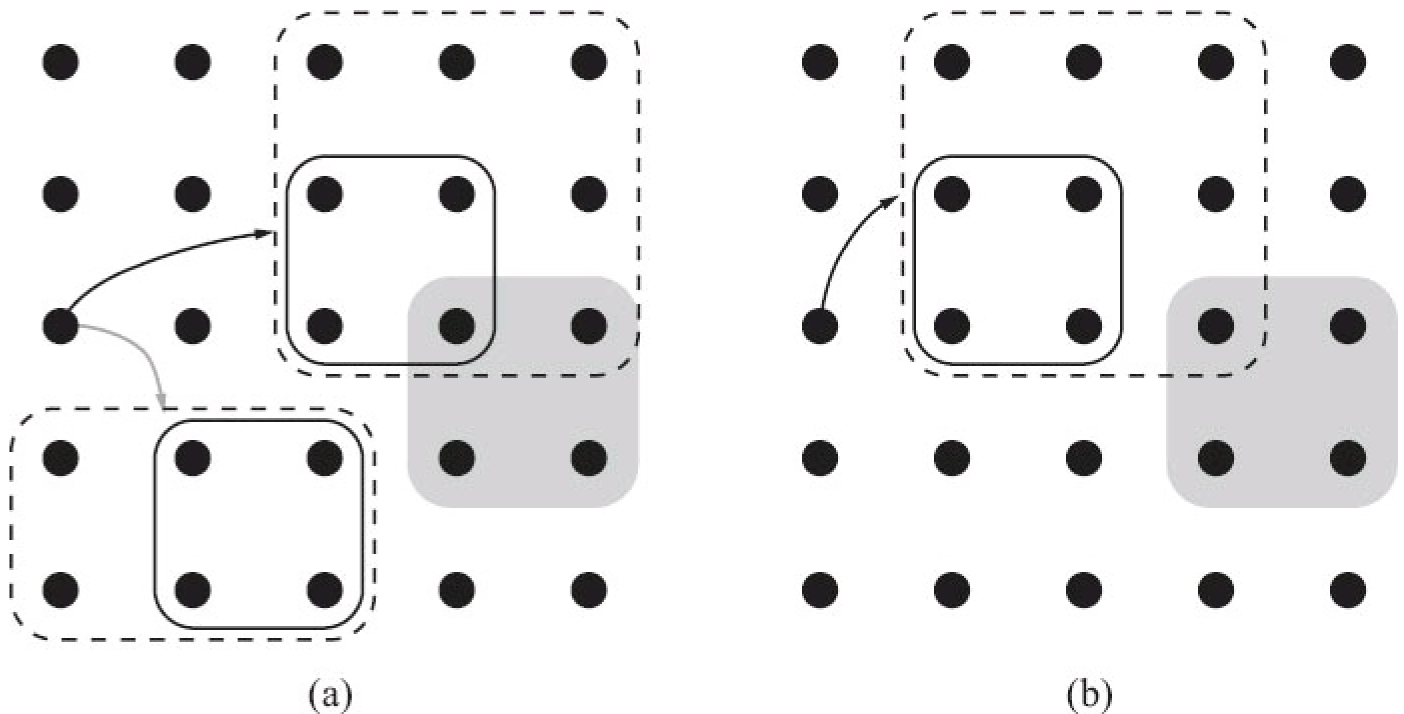


Figura 11.7 Realização do objetivo de planos de alto nível com descrições aproximadas. O conjunto de estados objetivo está sombreado. Para cada plano, são mostrados os conjuntos alcançáveis pessimista (linhas sólidas) e otimista (linhas tracejadas). (a) O plano indicado pela seta preta alcança o objetivo definitivamente, enquanto o plano indicado pela seta cinza, não. (b) Um plano que teria de ser refinado ainda mais para determinar se realmente atinge o objetivo.

Um algoritmo para o planejamento hierárquico com descrições angélicas aproximadas é mostrado na Figura 11.8. Para simplificar, mantivemos o mesmo esquema utilizado anteriormente na Figura 11.5, ou seja, uma busca em largura no espaço de refinamentos. Como explicado, o algoritmo pode detectar planos que vão ou não funcionar, marcando as interseções dos conjuntos alcançáveis otimistas e pessimistas com o objetivo (os detalhes de como calcular os conjuntos alcançáveis de um plano, dadas as descrições aproximadas de cada passo, são abordados no Exercício 11.5). Quando

se encontra um plano abstrato viável, o algoritmo *decompõe* o problema original em subproblemas, um para cada passo do plano. O estado inicial e objetivo de cada subproblema são obtidos regredindo um estado objetivo garantidamente alcançável através de esquemas de ação para cada passo do plano (veja a Seção 10.2.2 para uma discussão de como funciona a regressão). A Figura 11.6(b) ilustra a ideia básica: o estado marcado por um círculo no lado direito é o estado objetivo garantidamente alcançável, e o estado marcado por um circulo no lado esquerdo é o objetivo intermediário obtido pela regressão do objetivo por meio da ação final.

função BUSCA-ANGÉLICA (*problema*, *hierarquia*, *PlanoInicial*) **retorna** solução ou *falha*
  fronteira ← fila FIFO com *PlanoInicial* como elemento único
  fechar laço
    se VAZIO? (*fronteira*) **então retornar** *falha*
    plano ← POP(*fronteira*) /* escolhe o lugar mais raso da *fronteira**/
    se ALCANCE$^+$(*problema*.ESTADO-INICIAL, *plano*) interseciona *problema*.OBJETIVO **entã**
      se *plano* é primitivo **então retornar** *plano* /* ALCANCE$^+$ é exato para planos primitivos */
      garantido ← ALCANCE$^-$(*problema*.ESTADO-INICIAL, *plano*) > *problema*.OBJETIVO
      se *garantido* ≠ { } e FAZER-PROGRESSO (*plano*, *PlanoInicial*), **então**
        EstadoFinal ← qualquer elemento de *garantido*
        retornar DECOMPOR(*hierarquia*, *problema*.ESTADO-INICIAL, *plano*, *estadoFinal*)
      hla ← algumas HLA no *plano*
      prefixo, *sufixo* ← subsequências de ações antes e depois da *hla* no *plano*
      para cada *sequência* **em** REFINAMENTOS (*hla*, *consequência*, *hierarquia*) **faça**
        fronteira ← INSERIR(CONCATENAR(*prefixo*, *sequência*, *sufixo*), *fronteira*)

---

função DECOMPOR(*hierarquia*, s_0, *plano*, s_f) **retornar** uma solução
  solução ← um plano vazio
  enquanto *plano* não está vazia **faça**
    ação ← REMOVER-ÚLTIMO (*plano*)
    si ← um estado em ALCANCE$^-$(s_0 *plano,*) tal que s_f ALCANCE$^-$ (s_i, *ação*)
    problema ← problema com ESTADO-INICIAL = s_i e OBJETIVO = s_f
    solução ← CONCATENAR(BUSCA-ANGÉLICA(problema, hierarquia, ação), solução)
    $s_f \leftarrow s_i$
  retornar *solução*

Figura 11.8 Um algoritmo de planejamento hierárquico que usa semântica angélica para identificar e se comprometer com planos de alto nível que funcionam, evitando planos de alto nível que não funcionam. O predicado FAZER-PROGRESSO realiza verificações para se certificar de que não estamos presos em uma regressão infinita de refinamentos. No nível superior, chama a BUSCA-ANGÉLICA com [*Ação*] como o *PlanoInicial*.

A capacidade de comprometer-se ou de rejeitar planos de alto nível pode dar à BUSCA-ANGÉLICA uma vantagem computacional significativa sobre a BUSCA-HIERÁRQUICA, que por sua vez pode ter uma grande vantagem sobre a antiga e simples BUSCA-EM-LARGURA. Considere,

por exemplo, a limpeza com um grande mundo do aspirador de pó composto de quartos retangulares ligados por corredores estreitos. Faz sentido ter uma HLA para *Navegar* (como mostrado na Figura 11.4) e um para *LimparQuartoTodo* (a limpeza do quarto poderia ser implementada com a aplicação repetida de outra HLA para limpar cada fileira). Desde que há cinco ações nesse domínio, o custo da BUSCA-EM-LARGURA cresce 5^d, onde d é o comprimento da menor solução (cerca de duas vezes o número total de quadrados); o algoritmo não pode gerenciar nem mesmo dois quartos 2 × 2. A BUSCA-HIERÁRQUICA é mais eficiente, mas ainda sofre com o crescimento exponencial porque tenta todas as formas de limpar que seja consistente com a hierarquia. A BUSCA-ANGÉLICA apresenta escala aproximadamente linear no número de quadrados — compromete-se com uma boa sequência de alto nível e poda as outras opções. Observe que limpar um conjunto de quartos, limpando um de cada vez não é um bicho de sete cabeças: é fácil para os seres humanos justamente por causa da estrutura hierárquica da tarefa. Quando consideramos como os seres humanos acham difícil resolver pequenos quebra-cabeças, como o quebra-cabeças de oito peças, parece que a capacidade dos seres humanos para resolver problemas complexos deriva em grande parte de sua habilidade em abstrair e decompor o problema para eliminar a análise combinatória.

A abordagem angélica pode ser estendida para encontrar soluções de menor custo por generalizar a noção de conjunto alcançável. Em vez de um estado ser alcançável ou não, ela tem um custo para a forma mais eficiente de chegar lá (o custo é ∞ para os estados inalcançáveis). As descrições otimistas e pessimistas limitam esses custos. Dessa forma, a busca angélica pode comprovadamente encontrar planos abstratos ótimos, sem considerar suas implementações. A mesma abordagem pode ser usada para obtenção de **algoritmos lookahead hierárquicos** eficazes para busca on-line, no estilo do LRTA*. Em alguns aspectos, tais algoritmos espelham-se em aspectos de deliberação humana em tarefas tais como o planejamento de férias no Havaí — a consideração de alternativas é inicialmente feita em um nível abstrato em escalas longas de tempo; algumas partes do plano são deixadas completamente abstratas até o tempo de execução, como a forma de passar dois dias preguiçosos em Molokai, enquanto outras partes são planejadas em detalhes, tais como os voos a serem tomados e a hospedagem a ser reservada — sem esses refinamentos, não há garantia da viabilidade do plano.

11.3 PLANEJAMENTO E AÇÃO EM DOMÍNIOS NÃO DETERMINÍSTICOS

Nessa seção, estenderemos o planejamento para lidar com ambientes parcialmente observáveis, não determinísticos e desconhecidos. O Capítulo 4 estendeu a busca de forma semelhante, e os métodos aqui também são similares: **planejamento sem sensores** (também conhecido como **planejamento em conformidade**) para ambientes sem observações, **planejamento de contingência** para ambientes parcialmente observáveis e não determinísticos; e **planejamento on-line** e **replanejamento** para ambientes desconhecidos.

Enquanto os conceitos básicos são os mesmos que no Capítulo 4, há também diferenças significativas. Principalmente porque os planejadores lidam com representações fatoradas em vez de representações atômicas. Isso afeta a maneira como representamos a capacidade do agente para a ação e observação e a forma como representamos os **estados de crença** — os conjuntos de estados

físicos possíveis em que o agente possa estar — para ambientes não observáveis e parcialmente observáveis. Podemos também tirar proveito de muitos dos métodos independentes de domínio dados no Capítulo 10 para calcular heurísticas de busca.

Considere o problema: dada uma cadeira e uma mesa, o objetivo é combiná-las — ter a mesma cor. No estado inicial temos duas latas de tinta, mas as cores da tinta e dos móveis são desconhecidos. Apenas a mesa está inicialmente no campo de visão do agente:

$$Início(Objeto\ (Mesa) \land Objeto(Cadeira) \land Lata(C_1) \land Lata(C_2) \land Vista(Mesa))$$

$$Objetivo(Cor(Cadeira, c) \land Cor(Mesa, c))$$

Há duas ações: remover a tampa de uma lata de tinta e pintar um objeto usando a tinta da lata aberta. Os esquemas de ação são simples, com uma exceção: agora permitimos precondições e efeitos para conter variáveis que não são parte da lista de variáveis da ação. Ou seja, *Tinta*(*x, lata*) não menciona a variável *c*, que representa a cor da tinta na lata. No caso totalmente observável, isso não é permitido — teríamos que denominar a ação *Pintar*(*x, lata, c*). Mas, no caso parcialmente observável, podemos ou não saber que cor está na lata (a variável *c* é universalmente quantificada, assim como todas as outras variáveis em um esquema de ação).

Ação(*RemoverTampa*(*lata*),
PRECOND: *Lata*(*lata*)
EFFECT: *Abrir*(*lata*))
Ação(*Pintar*(*x, lata*),
PRECOND: *Object*(*x*) ∧ *Lata*(*lata*) ∧ *Cor*(*lata, c*) ∧ *Abrir*(*lata*)
EFFECT: *Cor*(*x, c*)).

Para resolver um problema parcialmente observável, o agente terá que raciocinar sobre as percepções que vai obter quando estiver executando o plano. A percepção será fornecida pelos sensores do agente quando estiver realmente em ação, mas quando estiver planejando será necessário um modelo de seus sensores. No Capítulo 4, esse modelo foi dado pela função, PERCEPÇÃO(*s*). Para o planejamento, que estende a PDDL com um novo tipo de esquema, o **esquema percepção**:

Percepção(Cor (x, c),
PRECOND: *Objeto(x)* ∧ *Vista(x)*
Percepção(Cor (lata, c),
PRECOND: *Lata(lata)* ∧ *Vista(lata)* ∧ *Abrir(lata)*

O primeiro esquema diz que, sempre que um objeto estiver à vista, o agente vai perceber a cor do objeto (isto é, para o objeto *x*, o agente vai aprender o valor verdadeiro da *Cor*(*x, c*) para todo *c*). O segundo esquema informa que, se uma lata aberta estiver à vista, o agente percebe a cor da tinta na lata. Como não há eventos exógenos nesse mundo, a cor de um objeto permanecerá a mesma, mesmo se não estiver sendo percebida, até que o agente execute uma ação para mudar a cor do objeto. Certamente o agente precisará de uma ação que faça com que os objetos (um de cada vez) fiquem à vista:

$Ação(Olhar(x),$
PRECOND: $Vista(y) \land (x \neq y)$
EFFECT: $Vista(x) \land \neg Vista(y))$.

Para um ambiente totalmente observável, teríamos um axioma *Percepção* sem precondições para cada fluente. Um agente sem sensores, por outro lado, não tem axiomas de *Percepção*. Observe que mesmo um agente sem sensores pode resolver o problema da pintura. Uma solução é abrir qualquer lata de tinta e aplicá-la tanto na cadeira como na mesa, **forçando-as** (coercing), assim, a ser da mesma cor (mesmo que o agente não saiba que cor é).

Um agente de planejamento contingente com sensores pode gerar um plano melhor. Primeiro, olha para a mesa e para a cadeira para obter as cores; se elas são as mesmas, o plano está feito. Se não, olha para as latas de tinta; se a tinta na lata for da mesma cor que uma peça do mobiliário, aplicará a tinta na outra peça. Caso contrário, pintará ambas as peças com qualquer cor.

Finalmente, um agente de planejamento on-line pode gerar um plano de contingência com menos ramificações a princípio — talvez ignorando a possibilidade que nenhuma tinta na lata combine com a mobília — e lide com os problemas à medida que surjam, replanejando-os. Também poderia lidar com erros de seus esquemas de ação. Considerando que um planejador de contingência simplesmente assume que os efeitos de uma ação sempre são bem-sucedidos — que a pintura da cadeira resolve —, um agente de replanejamento verificaria o resultado e faria um plano adicional para corrigir qualquer falha inesperada, tal como uma área sem pintura ou a cor original ainda a mostra.

No mundo real, os agentes usam uma combinação de abordagens. Fabricantes de automóveis vendem pneus sobressalentes e *airbags*, que são incorporações físicas de ramificações de planos de contingência projetados para lidar com pneus furados ou batidas de carro. Por outro lado, a maioria dos motoristas nunca considera essas possibilidades; quando surge um problema, respondem como agentes de replanejamento. Em geral, os agentes planejam apenas para contingências que tenham consequências importantes e uma chance não negligenciável de acontecer. Assim, um motorista de carro contemplando uma viagem através do deserto do Saara deve fazer planos de contingência explícitos para falhas, enquanto uma ida ao supermercado exige menos planejamento antecipado. A seguir, veremos cada uma das três abordagens em mais detalhes.

11.3.1 Planejamento sem sensores

A Secção 4.4.1 introduziu a ideia básica de busca em espaço de estado de crença para encontrar uma solução para os problemas sem sensores. A conversão de um problema de planejamento sem sensores para um problema de planejamento de estado de crença funciona da mesma maneira como n a Seção 4.4.1; as principais diferenças são que o modelo de transição física subjacente é representado por uma coleção de esquemas de ação e o estado de crença pode ser representado por uma fórmula lógica em vez de um conjunto de estados explicitamente enumerados. Para simplificar, vamos supor que o problema de planejamento subjacente seja determinístico.

O estado de crença inicial para o problema de pintura sem sensores pode ignorar os fluentes *Vista* porque o agente não tem sensores. Além disso, tomamos como dados os fatos imutáveis

Objeto(*Mesa*) ∧ *Objeto*(*Cadeira*) ∧ *Lata*(C_1) ∧ *Lata*(C_2) porque são válidos em cada estado de crença. O agente não sabe as cores das latas ou objetos, ou se as latas estão abertas ou fechadas, mas sabe que os objetos e as latas têm cores: $\forall x\ \exists c\ Cor\ (x, c)$. Depois de skolemizar (veja a Seção 9.5), obtemos o estado de crença inicial:

$$b_0 = Cor(x, C(x)).$$

No planejamento clássico, onde se faz a **suposição do mundo fechado**, teríamos que assumir que qualquer fluente não mencionado em um estado é falso, mas no planejamento sem sensores (e parcialmente observável) temos que mudar para uma **suposição do mundo aberto**, no qual os estados contêm fluentes positivos e negativos, e, se um fluente não aparecer, seu valor é desconhecido. Assim, o estado de crença corresponde exatamente ao conjunto de mundos possíveis que satisfazem a fórmula. Dado esse estado de crença inicial, a sequência de ação seguinte é uma solução:

$$[RemoverTampa(Lata_1)\ Pintar(Cadeira, Lata_1), Pintar(Mesa, Lata_1)].$$

Vamos agora mostrar o progresso do estado de crença através da sequência de ação para mostrar que o estado de crença final satisfaz o objetivo.

Primeiro, observe que, em determinado estado de crença *b*, o agente pode considerar qualquer ação cujas precondições sejam satisfeitas por *b* (as demais ações não podem ser utilizadas porque o modelo de transição não define os efeitos das ações cujas precondições não possam ser satisfeitas). De acordo com a Equação 4.4, a fórmula geral para atualizar o estado de crença *b* dada uma ação aplicável *a* em um mundo determinístico é a seguinte:

$$b' = \text{RESULTADO}(b, a) = \{s' : s' = \text{RESULTADO}_p(s, a) \text{ e } s \in b\}$$

onde RESULTADO_P define o modelo de transição física. Por enquanto, vamos supor que o estado de crença inicial seja sempre uma conjunção de literais, isto é, uma fórmula 1-FNC. Para construir o novo estado de crença *b'*, devemos considerar o que acontece com cada literal *l* em cada estado físico *s* em *b* quando a ação *a* for aplicada. Para literais cujo valor verdadeiro já é conhecido em *b*, o valor verdadeiro em *b'* é calculado a partir do valor atual e da lista de adição e exclusão da ação (por exemplo, se *l* estiver na lista de exclusão da ação, então $\neg l$ é adicionado a *b'*). E o que aconteceria com um literal cujo valor verdadeiro é desconhecido em *b*? Há três casos:

1. Se a ação adiciona *l*, então *l* será verdadeiro em *b'*, independentemente do seu valor inicial.
2. Se a ação exclui *l*, então *l* será falso em *b'*, independentemente do seu valor inicial.
3. Se a ação não afeta *l*, então *l* vai manter o seu valor inicial (que é desconhecido) e não aparecerá em *b'*.

Assim, vemos que o cálculo de *b'* é quase idêntico ao caso observável, que foi especificado pela Equação 10.1:

$$b' = \text{RESULTADO}(b, a) = (b - \text{DEL}\ (a)) \cup \text{ADD}\ (a).$$

Quase não podemos utilizar o conjunto semântico porque (1) temos de nos certificar de que *b'* não contém *l* ou ¬*l* e (2) de que os átomos podem conter variáveis livres. Mas ainda o caso é que o RESULTADO(*b*, *a*) é calculado começando com *b*, definindo qualquer átomo que apareça em DEL(*a*) como falso e definindo qualquer átomo que apareça em ADD(*a*) como verdadeiro. Por exemplo, se aplicarmos RemoverTampa(Lata1) para o estado de crença inicial b0, obteremos

$$b_1 = Cor(x, C(x)) \wedge Abrir(Lata_1).$$

Quando aplicamos a ação $Pintar(Cadeira, Lata_1)$, a precondição $Cor(Lata_1, c)$ é satisfeita pelo literal conhecido $Cor(x, C(x))$ com substituição $\{x/Lata_1, c / C(Lata_1)\}$ e o novo estado de crença é

$$b_2 = Cor(x, C(x)) \wedge Abrir(Lata_1) \wedge Cor(Cadeira, C(Lata_1)).$$

Finalmente, aplicamos a ação $Pintar(Mesa, Lata_1)$ para obter

$$b_3 = Cor(x, C(x)) \wedge Abrir(Lata_1) \wedge Cor(Cadeira, C(Lata_1)) \wedge Cor(Mesa, \mathrm{C}(Lata_1)).$$

O estado de crença final satisfaz o objetivo, $Cor(Mesa, c) \wedge Cor(Cadeira, c)$, com a variável *c* limitada a $C(Lata_1)$.

A análise anterior da regra de atualização mostrou um fato muito importante: *a família dos estados de crença definida como conjunções de literais é fechada sob as atualizações definidas pelos esquemas de ações PDDL*. Ou seja, se o estado de crença começar como uma conjunção de literais, qualquer atualização produzirá uma conjunção de literais. Isso significa que, em um mundo com *n* fluentes, qualquer estado de crença pode ser representado por uma conjunção de tamanho $O(n)$. Esse é um resultado muito reconfortante, considerando que existem 2^n estados no mundo. Isso informa que podemos representar compactamente todos os subconjuntos desses 2^n estados que vamos precisar. Além disso, o processo de verificação dos estados de crença que são subconjuntos ou superconjuntos de estados de crença visitados anteriormente também é fácil, pelo menos no caso proposicional.

O único senão desse quadro favorável é que só funciona para os esquemas de ação que têm os *mesmos efeitos* para todos os estados em que suas precondições são satisfeitas. É essa propriedade que permite a preservação da representação do estado de crença 1-FNC. Uma vez que o efeito depende do estado, as dependências são introduzidas entre os fluentes, e a propriedade 1-FNC é perdida. Considere, por exemplo, o mundo simples do aspirador de pó definido na Seção 3.2.1. Faça os fluentes AtL e AtR para a localização do robô e *LimpoL* e *LimpoR* para o estado dos quadrados. Segundo a definição do problema, a ação *Aspirar* não tem precondição — pode sempre ser feita. A dificuldade é que seu efeito depende da localização do robô: quando o robô está em AtL, o resultado é *LimpoL*, mas quando está em *AtR*, o resultado é *LimpoR*. Para essas ações, nossos esquemas de ação vão precisar de algo novo: um **efeito condicional**. A sintaxe é "**quando** *condição*: efeito", onde *condição* é uma fórmula lógica a ser comparada com o estado atual e *efeito* é uma fórmula que descreve o estado resultante. Para o mundo do aspirador de pó, temos

Ação(*Aspirar,*
EFEITO: **quando** *AtL*: *LimpoL* ∧ **quando** *AtR*: *LimpoR*).

Quando for aplicado ao estado de crença inicial *Verdadeiro*, o estado de crença resultante é (*AtL* ∧ *LimpoL*) ∨ (*AtR* ∧ *LimpoR*), que não é mais uma 1-FNC (essa transição pode ser vista na Figura 4.14). Em geral, os efeitos condicionais podem induzir a dependências arbitrárias entre os fluentes em um estado de crença, levando a estados de crença de tamanho exponencial no pior caso.

É importante entender a diferença entre precondições e efeitos condicionais. *Todos* os efeitos condicionais, cujas condições são satisfeitas, têm os seus efeitos aplicados para gerar o estado resultante; se nenhum for satisfeito, o estado resultante permanece inalterado. Por outro lado, se uma *precondição* não for satisfeita, a ação é inaplicável e o estado resultante é indefinido. Do ponto de vista do planejamento sem sensores, é melhor ter efeitos condicionais que uma ação inaplicável. Por exemplo, poderíamos dividir *Aspirar* em duas ações com efeitos incondicionais, da seguinte forma:

Ação(*AspirarL,*
PRECOND: *AtL*; EFEITO: *LimpoL*)
Ação(*AspirarR*,
PRECOND: *AtR*; EFEITO: *LimpoR*).

Agora temos apenas esquemas incondicionais, por isso todos os estados de crença permanecem em 1-FNC, mas, infelizmente, não podemos determinar a aplicabilidade de *AplicarL* e *AplicarR* no estado de crença inicial.

Parece inevitável, então, que os problemas não triviais vão envolver estados de crença sinuosos, como aqueles encontrados quando se considerou o problema de estimativa de estado para o mundo de wumpus (consulte a Figura 7.21). A solução então sugerida foi utilizar **aproximação conservativa** para o estado de crença exato, por exemplo, o estado de crença pode permanecer em 1-FNC se contiver todos os literais cujos valores verdadeiros possam ser determinados e tratar todos os outros literais como desconhecidos. Embora essa abordagem seja *boa*, por nunca gerar um plano incorreto, é i*ncompleta*, pois pode ser incapaz de encontrar soluções para problemas que envolvem necessariamente interações entre literais. Para dar um exemplo corriqueiro, se o objetivo for que o robô esteja em um quadrado limpo, então [*Aspirar*] é uma solução, mas um agente sem sensor que insiste em encontrar um estado de crença 1-FNC, não vai encontrá-lo.

Talvez a melhor solução seja procurar sequências de ações que mantenham o estado de crença o mais simples possível. Por exemplo, no mundo do aspirador de pó sem sensor, a sequência de ação [*Direita*, *Aspirar*, *Esquerda*, *Aspirar*] gera a seguinte sequência de estados crença:

$b_0 = Verdadeiro$
$b_1 = AtR$
$b_2 = AtR \land LimpoR$
$b_3 = AtL \land LimpoR$
$b_4 = AtL \land LimpoR \land LimpoL.$

Ou seja, o agente *pode* resolver o problema enquanto mantém um estado de crença 1-FNC, embora algumas sequências (por exemplo, as que começam com *Aspirar*) saiam da forma 1-FNC. A lição geral não foi perdida sobre os seres humanos: estamos sempre realizando pequenas ações (verificando a hora, batendo nos bolsos para ter certeza de que temos as chaves do carro, lendo placas de rua à medida que rodamos pela cidade) para eliminar a incerteza e manter o nosso estado de crença gerenciável.

Há uma outra abordagem bem diferente para o problema de estados de crença sinuosos: não é necessário calculá-los. Suponha que o estado de crença inicial seja b_0 e que gostaríamos de conhecer o estado de crença resultante da sequência de ação $[a_1,\ldots, a_m]$. Em vez de calculá-lo explicitamente, apenas represente-o como "b_0 então $[a_1, \ldots , a_m]$." Essa é uma representação preguiçosa mas inequívoca do estado de crença, e é bastante concisa — $O(n + m)$, onde n é o tamanho do estado de crença inicial (que se assume estar em 1-FNC) e m é o tamanho máximo de uma sequência de ações. Como uma representação do estado de crença ela sofre de uma desvantagem: determinar se o objetivo foi satisfeito ou se uma ação é aplicável pode exigir muito cálculo.

O cálculo pode ser implementado como um teste de consequência lógica: se A_m representa a coleção dos axiomas de estado sucessor necessária para definir as ocorrências das ações $a_1,\ldots\ a_m$ — como foi explicado para o SATPLAN na Seção 10.4.1 — e G_m afirma que o objetivo é verdadeiro depois de m etapas, então o plano atingirá o objetivo se $b_0 \wedge A_m \models G_m$, isto é, se $b_0 \wedge A_m \wedge \neg G_m$ for insatisfatível. Dado um solucionador moderno SAT, pode ser possível fazer isso muito mais rapidamente do que pelo cálculo do estado de crença total. Por exemplo, se nenhuma das ações na sequência tiver um fluente objetivo em particular na sua lista de adição, o solucionador vai detectar isso imediatamente. Também ajuda se os resultados parciais sobre o estado de crença — por exemplo, fluentes que se sabe serem verdadeiros ou falsos — são armazenados em cache para simplificar os cálculos subsequentes.

A última peça do quebra-cabeça de planejamento sem sensor é uma função heurística para orientar a busca. O significado da função heurística é o mesmo que no planejamento clássico: uma estimativa (talvez admissível) do custo de alcançar o objetivo de um determinado estado de crença. Com estados de crença, temos um fato adicional: resolver qualquer subconjunto de um estado de crença é necessariamente mais fácil do que resolver o estado de crença:

$$\text{se } b_1 \subseteq b_2 \text{ então } h^*(b_1) \leq h^*(b_2)$$

Portanto, qualquer heurística admissível calculada para um subconjunto é admissível para o estado de crença em si. Os candidatos mais óbvios são os subconjuntos unitários, ou seja, estados físicos individuais. Podemos tomar qualquer conjunto aleatório de estados $s_1,\ldots, s_N$ que estiverem no estado de crença b, aplicar qualquer heurística admissível h do Capítulo 10 e devolver

$$H(b) = \max \{h(s_1),\ldots, h(s_N)\}$$

como estimativa heurística para resolver b. Poderíamos também utilizar um grafo de planejamento diretamente sobre o próprio b: se for uma conjunção de literais (1-FNC), basta definir os literais para ser a camada de estado inicial do grafo. Se b não estiver em 1-FNC, pode ser possível

encontrar conjuntos de literais que, juntos, tem como consequência lógica *b*. Por exemplo, se *b* estiver na forma normal disjuntiva (FND), cada termo da fórmula FND será uma conjunção de literais que permite *b* e pode formar a camada inicial de um grafo de planejamento. Como antes, podemos tirar o máximo das heurísticas obtidas de cada conjunto de literais. Podemos utilizar também heurísticas inadmissíveis, como a heurística de ignorar a lista de exclusão, que parece funcionar muito bem na prática.

11.3.2 Planejamento contingente

Vimos no Capítulo 4 que o planejamento contingente — a geração de planos com ramificação condicional baseada em percepções — é apropriado para ambientes com observabilidade parcial, não determinismo ou ambos. Para o problema da tinta parcialmente observável com os axiomas de percepção, dado anteriormente, a solução de contingência possível é a seguinte:

[*OlharPara*(*Mesa*), *OlharPara*(*Cadeira*),
se *Cor*(*Mesa*, *c*) ∧ *Cor*(*Cadeira*, *c*), **então** *NoOp*
senão [*RemoverTampa*($Lata_1$), *OlharPara*($Lata_1$), *RemoverTampa*($Lata_2$), *OlharPara*($Lata_2$),
se *Cor*(*Mesa*, *c*) ∧ *Cor*(*lata*, *c*), **então** *Pintar*(*Cadeira*, *lata*)
senão se *Cor*(*Cadeira*, *c*) ∧ *Cor*(*lata*, *c*), **então** *Pintar*(*Mesa*, *lata*)
senão [*Pintar*(*Cadeira*, $Lata_1$), *Paint*(*Mesa*, $Lata_1$)]]]

As variáveis desse plano devem ser consideradas existencialmente quantificadas; a segunda linha informa que, se existe alguma cor *c* que seja a cor da mesa e da cadeira, então o agente não precisa fazer nada para atingir o objetivo. Ao executar esse plano, um agente de planejamento contingente pode manter o seu estado de crença como uma fórmula lógica e avaliar cada condição de ramificação determinando se o estado de crença permite a fórmula de condição ou sua negação. (É responsabilidade do algoritmo de planejamento contingente certificar--se de que o agente nunca termine em um estado de crença onde a condição do valor verdadeiro da fórmula seja desconhecido.) Observe que, com as condições de primeira ordem, a fórmula pode ser satisfeita de mais de uma maneira; por exemplo, a condição *Cor*(*Mesa, c*) ∧ *Cor*(*lata, c*) pode ser satisfeita por $\{lata/Lata_1\}$ e por $\{lata/Lata_2\}$ se ambas as latas forem da mesma cor que a mesa. Nesse caso, o agente pode escolher qualquer substituição satisfatória para aplicar ao resto do plano.

Conforme mostrado na Seção 4.4.2, calcular o novo estado de crença após uma ação e subsequente percepção, é feito em duas etapas. A primeira etapa calcula o estado de crença após a ação, assim como para o agente sem sensor:

$$\hat{b} = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

onde, como antes, assumimos um estado de crença representado como uma conjunção de literais. A segunda etapa é um pouco mais complicada. Suponha que os literais de percepção $p_1, \ldots, p_k$ sejam recebidos. Pode-se pensar que seja simplesmente necessário adicioná-los ao estado de crença; de fato, pode-se também inferir que as precondições de atividade sensorial estão satisfeitas. Agora, se

uma percepção *p* tem exatamente um axioma de percepção, *Percepção*(*p*, PRECOND:*c*), onde *c* é uma conjunção de literais, então esses literais podem ser jogados no estado de crença juntamente com *p*. Por outro lado, se *p* tiver mais que um axioma de percepção cujas precondições podem ser válidas de acordo com o estado de crença predito $\hat{b}$, então teremos que adicioná-los na *disjunção* das precondições. Obviamente, isso leva o estado de crença não estar em 1-FNC e traz à tona as mesmas complicações dos efeitos condicionais, com as mesmas classes de soluções.

Dado um mecanismo para o cálculo exato ou aproximado dos estados de crença, podemos gerar planos contingentes, com uma extensão da busca para a frente E-OU sobre os estados de crença utilizados na Seção 4.4. Ações com efeitos não determinísticos — que são definidos simplesmente usando uma disjunção no EFEITO do esquema de ação — podem ser acomodados com pequenas alterações para o cálculo de atualização do estado de crença e nenhuma mudança para o algoritmo de busca.[2] Para a função heurística, muitos dos métodos sugeridos para o planejamento sem sensor também são aplicáveis nos casos parcialmente observáveis, não determinísticos.

11.3.3 Replanejamento on-line

Imagine assistir a um robô de soldagem por pontos em uma fábrica de automóveis. Os movimentos rápidos e precisos do robô são repetidos inúmeros vezes à medida que cada carro passa pela linha. Embora tecnicamente impressionante, o robô provavelmente não parece de todo *inteligente* porque o movimento é uma sequência fixa, pré-programada; o robô obviamente não "sabe o que está fazendo" em qualquer sentido significativo. Agora, suponha que uma porta mal colocada caia do carro quando o robô está prestes a aplicar um ponto de solda. O robô rapidamente substitui seu atuador de solda com uma pinça, pega a porta, verifica se há arranhões, recoloca-a no carro, envia um e-mail para o supervisor da fábrica, volta para o atuador de solda e retoma seu trabalho. De repente, o comportamento do robô parece *intencional*, em vez de mecânico; assumimos que ele não resulta de um plano contingente amplo e pré-calculado, mas de um processo de replanejamento on-line — o que significa que o robô precisa saber o que está tentando fazer.

O replanejamento pressupõe alguma forma de monitoramento de execução para determinar a necessidade de um plano novo. Tal necessidade surge quando um agente de planejamento se cansa de planejar todas as pequenas contingências, como se o céu pudesse cair em sua cabeça.[3] Algumas ramificações de um plano contingente parcialmente construído podem dizer simplesmente *Replanejar*; se tal ramificação for atingida durante a execução, o agente reverte para o modo de planejamento. Como mencionamos anteriormente, a decisão sobre o quanto o problema deve ser resolvido por antecipação e o quanto deve ser deixado para replanejamento é a que envolve negociação entre os eventos possíveis com custos diferentes e probabilidade de ocorrer. Ninguém quer que seu carro quebre no meio do deserto do Saara para só então pensar em ter água suficiente.

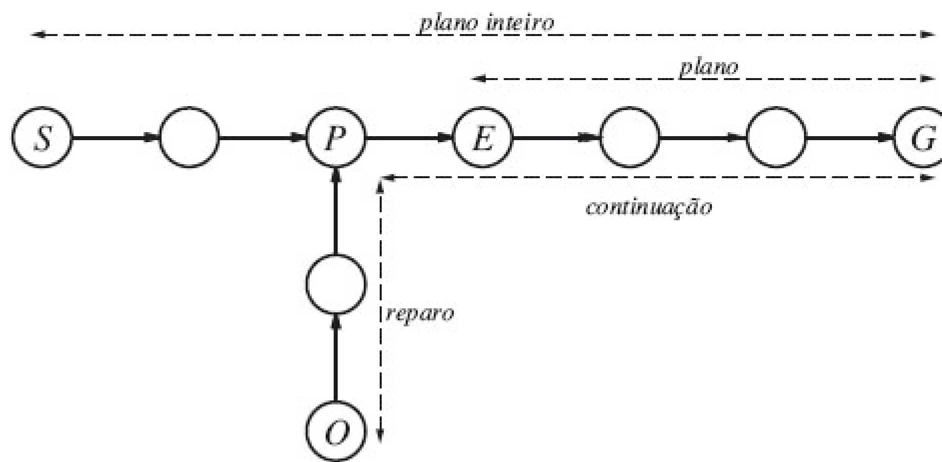


Figura 11.9 Antes da execução, o planejador vem com um plano, chamado de *plano total*, para ir de *S* para *G*. O agente executa as etapas do plano até que espera estar no estado *E*, mas observa que na verdade está em *O*. O agente então replaneja o reparo mínimo mais a continuação para alcançar *G*.

O replanejamento também pode ser necessário se o modelo do mundo do agente estiver incorreto. O modelo de uma ação pode ter uma **precondição ausente** — por exemplo, o agente pode não saber que muitas vezes é necessário uma chave de fenda para remover a tampa de uma lata de tinta; o modelo pode ter um **efeito ausente** — por exemplo, a pintura de um objeto pode também significar tinta no chão; ou o modelo pode ter uma **variável de estado ausente** — por exemplo, o modelo dado anteriormente não tinha noção da quantidade de tinta em uma lata, de como suas ações afetam esse valor ou da necessidade da quantidade ser diferente de zero. O modelo pode também não ter provisão para **eventos exógenos**, como alguém derrubar a lata de tinta. Os eventos exógenos podem também incluir mudanças no objetivo, como a adição do requisito de que a mesa e a cadeira não sejam pintadas de preto. Sem a capacidade de monitorar e replanejar, o comportamento de um agente é suscetível de ser extremamente frágil se confiar na correção absoluta de seu modelo.

O agente on-line tem a escolha de como monitorar o ambiente cuidadosamente. Distinguimos três níveis:

- **Monitoramento da ação**: antes de executar uma ação, o agente verifica se todas as precondições são válidas.
- **Monitoramento do plano**: antes de executar uma ação, o agente verifica se o restante do plano ainda vai ter sucesso.
- **Monitoramento do objetivo**: antes de executar uma ação, o agente verifica se há um conjunto melhor de objetivos que poderia tentar alcançar.

Na Figura 11.9, observamos um esquema de monitoramento de ações. O agente mantém o controle de ambos os planos originais, o *plano total*, e a parte do plano que não foi executada ainda, a qual é indicada por *plano*. Depois de executar as primeiros passos do plano, o agente espera estar no estado *E*. Mas o agente observa que está na verdade no estado *O*. Então precisa reparar o plano encontrando algum ponto *P* no plano original para o qual possa voltar (pode ser que *P* seja o estado objetivo, *G*). O agente tenta minimizar o custo total do plano: a parte de reparo (de *O* para *P*) além da continuação (de *P* para *G*).

Agora vamos voltar para o problema do exemplo de conseguir uma cadeira e mesa da mesma cor.

Suponha que o agente venha com esse plano:

[*OlharPara* (*Mesa*), *OlharPara* (*Cadeira*),
se *Cor* (*Mesa, c*) $\wedge$ *Cor* (*Cadeira, c*), **então** *NoOp*
senão [*RemoverTampa* ($Lata_1$), *OlharPara* ($Lata_1$),
se *Cor* (*Mesa, c*) $\wedge$ *Cor* ($Lata_1$, *c*), **então** *Pintar* (*Cadeira*, $Lata_1$)
senão REPLANEJAR.]]

Agora, o agente está pronto para executar o plano. Suponha que o agente observe que a mesa e a lata de tinta são brancos e a cadeira é preta. Então executa *Pintar*(*Cadeira*, $Lata_1$). Nesse ponto, um planejador clássico declararia sucesso, o plano foi executado. Mas uma execução on-line monitorando o agente precisa verificar as precondições dos planos vazios restantes — que a mesa e a cadeira são da mesma cor. Suponha que o agente perceba que elas não têm a mesma cor — de fato, a cadeira agora tem um cinza manchado porque a tinta preta está aparecendo. O agente, então, precisa descobrir uma posição no *plano total* a atingir e uma sequência de ações de reparo para chegar lá. O agente percebe que o estado atual é idêntico à precondição antes da ação *Pintar*(*Cadeira*, $Lata_1$), então escolhe uma sequência vazia para *reparo* e faz com que seu *plano* seja a mesma sequência [*Pintura*] que ele tinha acabado de tentar. Com esse novo plano em prática, reinicia o controle da execução, e a ação *Pintar* é repetida. Esse comportamento se repetirá até que seja percebido que a cadeira está totalmente pintada. Mas note que o laço foi criado por um processo de planejar-executar-replanejar, em vez de por um laço explícito em um plano. Observe também que o plano original não precisa tratar cada contingência. Se o agente atingir o passo marcada como REPLANEJAR, ele pode gerar um novo plano (talvez envolvendo $Lata_2$).

O monitoramento de ações é um método simples de monitoramento de execução, mas às vezes pode conduzir a um comportamento menos inteligente. Por exemplo, suponha que não haja tinta preta ou branca, e o agente construa um plano para resolver o problema da pintura pintando a mesa e a cadeira de vermelho. Suponha que haja apenas tinta vermelha suficiente para a cadeira. Com o monitoramento de ação, o agente poderia ir em frente e pintar a cadeira de vermelho; em seguida, perceberia que a tinta tinha acabado e não poderia pintar a mesa, nesse ponto teria que replanejar um reparo — talvez pintando tanto a cadeira como a mesa de verde. Um agente de monitoramento de plano pode detectar falha sempre que o estado atual é tal que o plano restante não funciona mais. Assim, não vai perder tempo pintando a cadeira de vermelho. O controle de plano consegue isso verificando o sucesso das precondições de todo o plano restante, isto é, as precondições de cada passo no plano, exceto as que são atingidas por outro passo do plano restante. O monitoramento do plano corta a execução de um plano condenado o mais rapidamente possível, em vez de continuar até que a falha realmente ocorra.[4] O monitoramento do plano também permite **descobertas ao acaso** — sucesso acidental. Se alguém chega e pinta a mesa de vermelho ao mesmo tempo em que o agente está pintando a cadeira de vermelho, então as precondições do plano final são satisfeitas (o objetivo foi alcançado), e o agente pode ir para casa mais cedo.

É fácil modificar um algoritmo de planejamento para que cada ação no plano seja anotada com as precondições da ação, possibilitando assim o monitoramento da ação. É um pouco mais complexo permitir o monitoramento do plano. Planejadores de ordem parcial e planejamento em grafo têm a vantagem de já ter construído estruturas que contêm as relações necessárias para o monitoramento do

plano. Os planejadores estendidos com anotações necessárias podem ser feitos com um armazenamento cuidadoso à medida que o fluente objetivo é regredido através do plano.

Agora que descrevemos um método para monitoramento e replanejamento, é necessário perguntar: "Isso funciona?" Essa é uma pergunta surpreendentemente complicada. Se quisermos dizer "Pode-se garantir que o agente sempre atingirá o objetivo?", a resposta é não, porque o agente poderia chegar inadvertidamente a um beco sem saída, do qual não há reparo. Por exemplo, o agente do aspirador de pó pode ter um modelo defeituoso de si mesmo e não saber que suas baterias podem esgotar-se. Uma vez esgotadas, não é possível reparar quaisquer planos. Se descartarmos os becos sem saída — suponha que exista um plano para alcançar o objetivo de *qualquer* estado no ambiente — e suponha ainda que o ambiente é realmente não determinístico, no sentido de que tal plano tem sempre *alguma* chance de sucesso em qualquer tentativa de execução dada, então eventualmente o agente vai atingir o objetivo.

O problema ocorre quando uma ação não é realmente não determinística, mas depende de alguma precondição que o agente não conhece. Por exemplo, às vezes, uma lata de tinta pode estar vazia, então a tinta dessa lata não tem efeito. Nenhuma quantidade de novas tentativas vai modificar isso.[5] Uma solução é escolher aleatoriamente entre o conjunto de planos de reparo possível, em vez de tentar o mesmo a cada vez. Nesse caso, o plano de reparo de abrir outra lata poderá funcionar. Uma abordagem melhor é **aprender** um modelo melhor. Toda previsão de falha é uma oportunidade de aprendizagem; um agente deveria ser capaz de modificar seu modelo do mundo de acordo com sua percepção. A partir de então, o replanejador será capaz de chegar a um reparo que chegue à raiz do problema, em vez de confiar na sorte para escolher um bom reparo. Esse tipo de aprendizagem será descrito nos Capítulos 18 e 19.

11.4 PLANEJAMENTO MULTIAGENTE

Até agora, assumimos que apenas um agente está fazendo a percepção, o planejamento e a ação. Quando existem múltiplos agentes no ambiente, cada agente enfrenta um **problema de planejamento multiagente** no qual tenta alcançar seus próprios objetivos com a ajuda ou o impedimento de outros.

Entre o agente puramente único e os casos verdadeiramente de multiagentes existe um amplo espectro de problemas que exibem vários graus de decomposição do agente monolítico. Um agente com múltiplos atuadores que podem operar simultaneamente — por exemplo, um humano que consegue digitar e falar ao mesmo tempo — necessita fazer **planejamento multi-atuador** para gerir cada atuador enquanto trata interações positivas e negativas entre os atuadores. Quando os atuadores estão dissociados fisicamente em unidades separadas — como em uma frota de robôs de entrega em uma fábrica —, o planejamento multi-atuador torna-se o **planejamento multicorpo**. Um problema multicorpo ainda é um problema de agente único "padrão", desde que as informações relevantes sensoriais obtidas por cada corpo possam ser agrupadas — centralmente ou em cada corpo — para formar uma estimativa comum do estado do mundo que então informa a execução do plano global; nesse caso, os múltiplos corpos atuam como um único corpo. Quando as restrições de comunicação tornam isso impossível, temos o que é chamado às vezes de problema de **planejamento descentralizado**; este é talvez um equívoco porque a fase de planejamento é centralizada, mas a fase

de execução é, pelo menos parcialmente, dissociada. Nesse caso, o subplano construído para cada corpo pode necessitar incluir ações de comunicação explícitas com outros corpos. Por exemplo, os robôs de reconhecimento múltiplo que cobrem uma área ampla podem estar muitas vezes fora de contato por rádio com os outros e devem compartilhar suas descobertas durante as vezes em que a comunicação é viável.

Quando uma única entidade está fazendo o planejamento, há realmente apenas um objetivo que todos os corpos compartilham necessariamente. Quando os corpos são agentes distintos que fazem seu próprio planejamento, eles ainda podem compartilhar objetivos idênticos, por exemplo, dois tenistas humanos que formam uma dupla compartilham o objetivo de ganhar a partida. Mesmo com objetivos comuns, no entanto, os casos de multicorpos e multiagentes são bastante diferentes. Em uma equipe de duplas multicorpo robótico, um plano único dita que corpo vai para qual lugar na quadra e que corpo vai bater na bola. Em times de duplas multiagentes, por outro lado, cada agente decide o que fazer; sem um método de **coordenação**, ambos os agentes podem decidir cobrir a mesma parte da quadra e cada um pode deixar a bola para o outro bater.

O caso mais claro de um problema multiagente, certamente, é quando os agentes têm objetivos diferentes. No tênis, os objetivos de dois times opostos estão em conflito direto, levando à situação de soma zero do Capítulo 5. Os espectadores poderão ser vistos como agentes se seu apoio ou indiferença for um fator significativo e puder ser influenciado pela conduta dos jogadores; caso contrário, eles podem ser tratados como um aspecto da natureza — assim como o clima —, que se presume ser indiferente às intenções dos jogadores.[6]

Finalmente, alguns sistemas são uma mistura de planejamento centralizado e multiagentes. Por exemplo, uma empresa de entrega pode fazer um planejamento centralizado e *off-line* para as rotas de seus caminhões e aviões a cada dia, mas deixar alguns aspectos abertos para decisões autônomas por motoristas e pilotos que possam responder individualmente às situações de tráfego e às condições meteorológicas. Além disso, os objetivos da empresa e de seus funcionários são ajustados, até certo ponto, mediante o pagamento de **incentivos** (salários e bônus) — um sinal claro de que esse é um sistema multiagente de verdade.

As questões envolvidas no planejamento multiagente podem ser divididas basicamente em dois conjuntos. O primeiro, abrangido na Seção 11.4.1, envolve questões de representação e planejamento para múltiplas ações simultâneas; esses problemas ocorrem em todas as configurações de planejamento multi-atuadores e multiagente. O segundo, abrangido na Seção 11.4.2, envolve questões de cooperação, coordenação e concorrência resultantes de ambientes multiagentes de verdade.

11.4.1 Planejamento com várias ações simultâneas

Por enquanto, vamos tratar as configurações multi-atuadores, multicorpo e multiagente da mesma forma, rotulando genericamente como configurações **multiatores**, utilizando o termo genérico **atores** para abranger atuadores, corpos e agentes. O objetivo desta seção é desenvolver como definir modelos de transição, planos corretos e algoritmos eficientes de planejamento para a configuração multiatores. Um plano correto é aquele que, se executado por atores, alcança o objetivo (certamente, na configuração multiagente verdadeira, os agentes não podem concordar em executar qualquer plano

particular, mas pelo menos sabem que planos funcionariam se concordassem em executá-los). Para simplificar, assumimos a **sincronização** perfeita: cada ação leva a mesma quantidade de tempo, e ações em cada ponto do plano conjunto são simultâneas.

Começamos com o modelo de transição; para o caso determinístico, essa é a função RESULTADO(s, a). No cenário de agente único, pode haver b escolhas diferentes para a ação; b pode ser muito grande, especialmente para as representações de primeira ordem com muitos objetos para influenciar; no entanto, os esquemas de ação fornecem uma representação concisa. No cenário multiatores com n atores, a única ação a é substituída por uma **ação conjunta** $<a_1,\ldots, a_n>$, onde a_i é a ação tomada pelo i-ésimo ator. Imediatamente, vemos dois problemas: primeiro, temos que descrever o modelo de transição para b^n diferentes ações conjuntas; em segundo lugar, temos um problema de planejamento conjunto com um fator de ramificação de b^n.

Ao colocar atores juntos em um sistema multiator com um enorme fator de ramificação, o foco principal da pesquisa sobre planejamento multiatores tem sido *desacoplar* os atores na medida do possível, para que a complexidade do problema cresça linearmente com n, em vez de exponencialmente. Se os atores **não têm interação** uns com os outros — por exemplo, n atores, cada um jogando paciência —, então podemos simplesmente resolver n problemas distintos. Se os atores forem de **baixo acoplamento,** podemos conseguir algo próximo a essa melhoria exponencial? Naturalmente, essa é uma questão central em muitas áreas de IA. Vimos isso explicitamente no contexto de CSPs, onde os grafos de restrição do "tipo árvore" renderam métodos de solução eficientes, bem como o contexto de bancos de dados-padrão disjuntos e a heurística aditiva de planejamento.

A abordagem-padrão para os problemas de baixo acoplamento é fingir que os problemas são completamente dissociados e, em seguida, arrumar as interações. Para o modelo de transição, isso significa escrever esquemas de ação, como se os atores atuassem de forma independente. Vamos ver como isso funciona para o problema das duplas de tênis. Vamos supor que, em um ponto do jogo, o objetivo da equipe seja devolver a bola que foi jogada para ela e garantir que pelo menos um deles estará cobrindo a rede.

A primeira passagem para uma definição multiatores poderia parecer como na Figura 11.10. Com essa definição, é fácil verificar que o seguinte **plano conjunto** funciona:

PLANO 1:
A : [*Ir* (*A*, *LinhaDeFundoDireita*), *Bater*(*A*, *Bola*)]
B : [*NoOp* (*B*), *NoOp* (*B*)].

Atores(*A, B*)
Início(*Em* (*A, LinhaDeFundoEsquerda*) ∧ *Em*(*B, RedeDireita*) ∧
Aproximando(*Bola, LinhaDeFundoDireita*)) ∧ *Parceiro*(*A, B*) ∧ *Parceiro*(*B, A*)
Objetivo(*Retornada*(*Bola*) ∧ (*Em*(*a, RedeDireita*) ∧ *Em*(*a, RedeEsquerda*))
Ação (*Bater*((*ator, Bola*)),
PRECOND: *Aproximando*(*Bola, loc*) ∧ *Em*(*ator, loc*)
EFEITO: *Devolvida*(*Bola*))
Ação(*Ir*(*ator, para*),

PRECOND: *Em*(*ator, loc*) $\wedge$ *para* $\neq$ *loc*,
EFEITO: *Em*(*ator, para*) $\wedge$ $\neg$*Em*(*ator, loc*))

Figura 11.10 O problema de duplas no tênis. Dois atores *A* e *B* estão jogando juntos e podem estar em um de quatro locais: *LinhaDeFundoEsquerda*, *LinhaDeFundoDireita*, *RedeEsquerda* e *RedeDireita*. A bola só poderá ser retornada se o jogador estiver no lugar certo. Note que cada ação deve incluir o ator como um argumento.

Entretanto, os problemas surgem quando um plano tem os dois agentes batendo na bola, ao mesmo tempo. No mundo real, isso não vai funcionar, mas o esquema de ação para *Bater* diz que a bola será devolvida com sucesso. Tecnicamente, a dificuldade é que as precondições restringem o *estado* em que uma ação pode ser executada com êxito, mas não restringem outras ações que podem atrapalhar. Resolvemos isso aumentando os esquemas de ação com um novo recurso: a **lista de ação concorrente** indicando quais ações devem ou não ser executadas simultaneamente. Por exemplo, a ação *Bater* poderia ser descrita da seguinte forma:

Ação(*Bater*(*a, Bola*),
  CONCORRENTE: b $\neq$ a $\Rightarrow$ $\neg$*Bater*(*b, Bola*)
  PRECOND: *Aproximando*(*Bola, loc*) $\wedge$ *Em*(*a, loc*)
  EFEITO: *Devolvida*(*Bola*)).

Em outras palavras, a ação *Bater* tem seu efeito expresso apenas se não ocorrer por outro agente outra ação *Bater* ao mesmo tempo (na abordagem SATPLAN, seria manipulado por um **axioma de exclusão de ação** parcial). Para algumas ações, o efeito desejado é alcançado *somente* quando outra ação ocorre simultaneamente. Por exemplo, são necessários dois agentes para transportar uma geladeira cheia de bebidas para a quadra de tênis:

Ação(*Transportar*(*a, refrigerador, aqui, lá*),
  CONCORRENTE: b $\neq$ a $\wedge$ *Transportar*(*b, refrigerador, aqui, lá*)
  PRECOND: *Em*(*a, aqui*) $\wedge$ *Em*(*refrigerador, aqui*) $\wedge$ *Refrigerador*(*refrigerador*)
  EFEITO: *Em*(*a, lá*) $\wedge$ *Em*(*refrigerador, lá*) $\wedge$ $\neg$*Em*(*a, aqui*) $\wedge$ $\neg$*Em*(*refrigerador, aqui*)).

Com esses tipos de esquemas de ação, qualquer um dos algoritmos de planejamento descritos no Capítulo 10 pode ser adaptado com apenas pequenas modificações para gerar planos multiatores. Na medida em que o acoplamento entre os subplanos está solto — o que significa que as restrições de concorrência são considerados apenas raramente durante a busca do plano —, espera-se que as várias heurísticas derivadas do planejamento de agente único sejam também eficazes no contexto multiator. Poderíamos estender essa abordagem com os refinamentos dos dois últimos capítulos — HTNs, observabilidade parcial, condicionais, monitoramento de execução e replanejamento —, mas isso está fora do escopo deste livro.

11.4.2 Planejamento com múltiplos agentes: cooperação e coordenação

Agora vamos considerar o verdadeiro cenário multiagente em que cada agente faz o seu próprio plano. Para começar, vamos supor que os objetivos e a base do conhecimento sejam compartilhados. Pode-se pensar que isso se reduz ao caso de multicorpo — cada agente simplesmente calcula a solução conjunta e executa sua própria parte dessa solução. Infelizmente, o "*a*" em "*a solução conjunta*" é enganoso. Para a nossa equipe de duplas, existe mais do que uma solução conjunta:

$$\begin{aligned}&\text{PLANO 2:}\\&\qquad A : [Ir(A, RedeEsquerda), NoOp(A)]\\&\qquad B : [Ir(B, LinhaDeFundoDireita), Bater(B, Bola)].\end{aligned}$$

Se ambos os agentes podem concordar com o plano 1 ou o plano 2, o objetivo será alcançado. Mas, se *A* escolher o plano 2 e *B* escolher o plano 1, ninguém vai devolver a bola. Por outro lado, se *A* escolher 1 e *B* escolher 2, ambos vão tentar acertar a bola. Os agentes podem perceber isso, mas como podem coordenar isso para se certificar de que concordam com o plano?

Uma opção é adotar uma **convenção** antes de se envolver em atividade conjunta. A convenção é qualquer restrição sobre a seleção de planos conjuntos. Por exemplo, a convenção "permanecer em um lado da quadra" descartaria o plano 1, fazendo com que os parceiros de duplas escolhessem o plano 2. Os motoristas em uma estrada enfrentam o problema de não colidir uns com os outros, o que é (parcialmente) resolvido adotando a convenção "ficar no lado direito da estrada"; na maioria dos países, a alternativa, "ficar no lado esquerdo" funciona igualmente bem desde que todos os agentes estejam de acordo em um ambiente. No desenvolvimento da linguagem humana aplicam-se considerações similares, onde o importante não é qual idioma cada indivíduo deve falar, mas o fato de toda uma comunidade falar o mesmo idioma. Quando as convenções são comuns, são chamadas de **leis sociais**.

Na ausência de uma convenção, os agentes podem utilizar a **comunicação** para atingir conhecimentos comuns de um plano conjunto viável. Por exemplo, um jogador de tênis poderia gritar "Meu!" ou "Seu!" para indicar a preferência de um plano conjunto. Vamos cobrir os mecanismos de comunicação com mais profundidade no Capítulo 22, onde observaremos que a comunicação não envolve necessariamente uma troca verbal. Por exemplo, um jogador pode comunicar a preferência de um plano conjunto com o outro simplesmente ao executar a primeira parte dele. Se o agente *A* for para a rede, o agente *B* será obrigado a ir para a linha de fundo para bater a bola porque o plano 2 é o único plano conjunto que começa com *A* indo para a rede. Essa abordagem de coordenação, às vezes chamada de **reconhecimento do plano**, funciona quando uma única ação (ou uma curta sequência de ações) é suficiente para determinar um plano conjunto de forma inequívoca. Observe que a comunicação pode funcionar tão bem com agentes competitivos como com os cooperativos.

As convenções podem também surgir através de processos evolutivos. Por exemplo, formigas ceifadeiras comedoras de sementes são criaturas sociais que evoluíram a partir das vespas, que são menos sociais. As colônias de formigas executam planos conjuntos muito elaborados sem qualquer controle centralizado — o trabalho da rainha é de reproduzir, não de fazer planejamento centralizado — e com cada formiga com capacidade limitada de cálculo, comunicação e memória (Gordon, 2000, 2007). A colônia tem muitos papéis, incluindo os trabalhadores do interior, patrulheiros e exploradores que buscam as sementes. Cada formiga escolhe desempenhar um papel de acordo com as condições locais que ela observa. Por exemplo, as exploradoras viajam para longe do ninho, em

busca de uma semente, e, quando encontram uma, trazem-na de volta imediatamente. Assim, a taxa pela qual as exploradoras retornam ao ninho é uma aproximação da disponibilidade de alimentos diária. Quando a taxa é alta, outras formigas abandonam sua função atual e assumem o papel de escavadeiras. As formigas parecem ter uma convenção sobre a importância dos papéis — o explorador é o mais importante —, e as formigas facilmente alternam nos papéis mais importantes, mas não nos menos importantes. Existe algum mecanismo de aprendizagem: uma colônia aprende a fazer ações mais bem-sucedidas e prudentes ao longo do curso de sua vida de décadas, embora as formigas individuais vivam apenas cerca de um ano.

Um último exemplo de comportamento multiagente cooperativo aparece no comportamento de um bando de aves. Podemos obter uma simulação razoável de um bando se cada agente ave (algumas vezes chamado de *boid*) observar as posições dos seus vizinhos mais próximos e depois escolher a direção e a aceleração que maximizam a soma ponderada desses três componentes:

1. Coesão: uma pontuação positiva por se aproximar da posição média dos vizinhos
2. Separação: uma pontuação negativa por chegar muito perto de qualquer vizinho
3. Alinhamento: uma pontuação positiva por se aproximar da direção média dos vizinhos

Se todos os *boids* executarem essa política, o bando apresentará o **comportamento emergente** de voar como um corpo pseudorrígido com densidade mais ou menos constante que não se dispersa ao longo do tempo e que, ocasionalmente, faz movimentos repentinos de arremesso. Você pode ver uma imagem na Figura 11.11(a) e compará-la com um rebanho real em (b). Tal como acontece com as formigas, não há necessidade de cada agente possuir um plano conjunto que modele as ações de outros agentes.

Os problemas multiagentes mais difíceis envolvem tanto a cooperação com os membros da própria equipe como a competição com os membros de equipes adversárias, tudo sem controle centralizado. Vemos isso em jogos, como futebol robótico ou no jogo NERO, mostrado na Figura 11.11(c), em que duas equipes de agentes de software competem para capturar as torres de controle. Os métodos de planejamento eficaz nesse tipo de ambientes — por exemplo, tirar vantagem de acoplamentos fracos — estão ainda engatinhando.

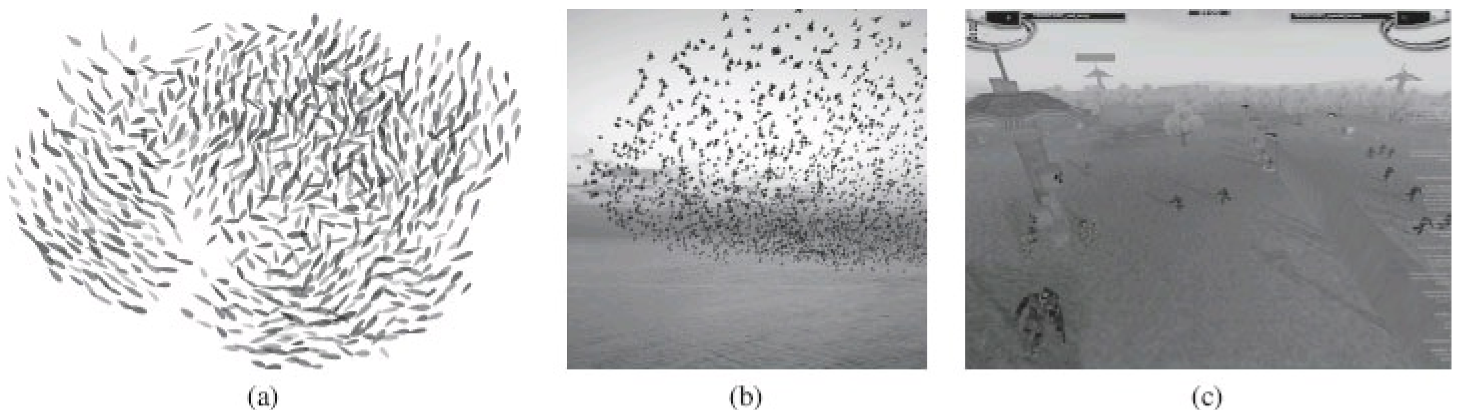


Figura 11.11 (a) Um bando de aves simulado, utilizando o modelo *boids* de Reynold. Cortesia da imagem de Giuseppe Randazzo, novastructura.net. (b) Um bando de estorninhos reais. Imagem de Eduardo (*pastaboy sleeps on a flickr*). (c) Duas equipes competitivas de agentes tentando capturar as torres no jogo NERO. Imagem cedida por Risto Miikkulainen.

11.5 RESUMO

Este capítulo examinou algumas das complicações do planejamento e da ação no mundo real. Os pontos mais importantes são:

- Muitas ações consomem **recursos**, como dinheiro, gás ou matérias-primas. É conveniente tratar esses recursos como medidas numéricas em um *pool*, em vez de tentar raciocinar, digamos, sobre cada moeda e cada cédula individual existente no mundo. As ações podem gerar e consumir recursos, e normalmente é econômico e eficiente verificar se os planos parciais satisfazem as restrições de recursos, antes de tentar realizar aprimoramentos adicionais.
- O tempo é um dos recursos mais importantes. Ele pode ser manipulado por algoritmos especializados de escalonamento ou o escalonamento pode ser integrado ao planejamento.
- O planejamento de **rede de tarefas hierárquicas** (HTN) permite ao agente receber conselhos do projetista de domínio, sob a forma de ações de alto nível (HLAs), que podem ser implementadas de várias formas por sequências de ações de baixo nível. Os efeitos das HLAs podem ser definidos com **semânticas angélicas**, permitindo provavelmente que sejam derivados planos corretos de alto nível sem considerar as implementações de mais baixo nível. Métodos HTN podem criar os planos realmente grandes exigidos por muitas aplicações do mundo real.
- Algoritmos de planejamento-padrão assumem informações completas e corretas em ambientes determinísticos totalmente observáveis. Muitos domínios violam essa suposição.
- Os **planos contingentes** permitem que o agente sinta o mundo durante a execução para decidir que ramo do plano seguir. Em alguns casos, o **planejamento sem sensor ou de conformidade** pode ser utilizado para construir um plano que funcione sem a necessidade de percepção. Tanto o plano de conformidade como de contingência podem ser construídos pela busca no espaço de **estados de crença**. A representação ou a computação eficiente dos estados de crença é um problema-chave.
- Um **agente de planejamento on-line** utiliza monitoramento de execução e reparos quando necessário para se recuperar de situações inesperadas que podem ser devidas a ações não determinísticas, eventos exógenos ou modelos incorretos do meio ambiente.
- O planejamento **multiagente** é necessário quando existem outros agentes no ambiente com os quais cooperar ou competir. Podem ser construídos planos conjuntos, mas devem ser aumentados com alguma forma de coordenação se dois agentes tiverem que concordar sobre qual plano conjunto executar.
- Este capítulo estende o planejamento clássico para abranger ambientes não determinísticos (em que os resultados das ações são incertos), mas não é a última palavra em planejamento. O Capítulo 17 descreve as técnicas para ambientes estocásticos (em que os resultados das ações têm probabilidades associadas a elas): processos markovianos de decisão, parcialmente observáveis e teoria dos jogos. No Capítulo 21, mostraremos que o aprendizado por reforço permite a um agente aprender como se comportar com os sucessos e fracassos do passado.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O planejamento com restrição de tempo foi tratado inicialmente pelo DEVISER (Vere, 1983). A representação de tempo em planos foi tratada por Allen (1984) e por Dean *et al*. (1990) no sistema FORBIN. O NONLIN+ (Tate e Whiter, 1984) e o SIPE (Wilkins, 1988, 1990) eram capazes de raciocinar sobre a alocação de recursos limitados para vários passos de plano. O O-PLAN (Bell e Tate, 1985), um planejador de HTN, tinha uma representação uniforme e geral para restrições sobre tempo e recursos. Além da aplicação da Hitachi mencionada no texto, o O-PLAN tem sido aplicado ao planejamento de compra de software na Price Waterhouse e ao planejamento da montagem do eixo traseiro dos automóveis da Jaguar Cars.

Os dois planejadores SAPA (Do e Kambhampati, 2001) e T4 (Haslum e Geffner, 2001) utilizam a busca direta em espaço de estados com heurísticas sofisticadas para manipular ações com durações de tempo e recursos. Uma alternativa é usar linguagens de ação muito expressivas, mas orientá-las por heurísticas específicas de domínios escritas por seres humanos, como foi feito no caso do ASPEN (Fukunaga *et al*., 1997), do HSTS (Jonsson *et al*., 2000) e do IxTeT (Ghallab e Laruelle, 1994).

Vários sistemas híbridos de planejamento e escalonamento foram implantados: o ISIS (Fox *et al*., 1982;. Fox, 1990) tem sido usado para escalonamento de linha de produção na Westinghouse, o GARI (Descotte e Latombe, 1985) planejou a construção de partes mecânicas, o Forbin foi usado para o controle da fábrica e o NONLIN+ foi usado para o planejamento de logística naval. Escolhemos apresentar planejamento e escalonamento como dois problemas distintos; Cushing *et al*. (2007) mostram que isso pode levar a incompletude sobre determinados problemas. Existe um longo histórico de escalonamento na indústria aeroespacial. O T-SCHED (Drabble, 1990) foi usado para escalonar sequências de comandos de missões para o satélite UOSAT-II. O OPTIMUM-AIV (Aarup *et al*., 1994) e o PLAN-ERS 1 (Fuchs *et al*., 1990), ambos baseados no O-PLAN, foram usados na montagem de naves espaciais e em planejamento de observação, respectivamente, na agência espacial europeia. O SPIKE (Johnston e Adorf, 1992) foi empregado para planejamento de observação na Nasa para o telescópio espacial Hubble, enquanto o Space Shuttle Ground Processing Scheduling System (Deale *et al*., 1994) realiza o escalonamento de linha de produção de até 16.000 deslocamentos de trabalhadores. O Remote Agent (Muscettola *et al*., 1998) se tornou o primeiro planejador-escalonador autônomo a controlar uma nave espacial quando voou a bordo da sonda Deep Space One em 1999. As aplicações espaciais impulsionaram o desenvolvimento de algoritmos para alocação de recursos; consulte Laborie (2003) e Muscettola (2002). A literatura sobre escalonamento é apresentada em um artigo de revisão clássico (Lawler *et al*., 1993), um livro recente (Pinedo, 2008) e em uma coletânea editada (Blazewicz *et al*., 2007).

A capacidade do programa STRIPS para aprendizado de **macrops** — “macro-operadores” que consistiam em uma sequência de passos primitivos — pode ser considerado o primeiro mecanismo para planejamento hierárquico (Fikes *et al*., 1972). A hierarquia também foi usada no sistema LAWALY (Siklossy e Dreussi, 1973). O sistema ABSTRIPS (Sacerdoti, 1974) introduziu a ideia de uma **hierarquia de abstração**, por meio da qual o planejamento em níveis mais altos podia ignorar precondições de ações de nível mais baixo com a finalidade de derivar a estrutura geral de um plano viável. A tese de doutorado de Austin Tate (1975b) e o trabalho de Earl Sacerdoti (1977) desenvolveram as ideias básicas do planejamento de HTN em sua forma moderna. Muitos planejadores práticos, incluindo o O-PLAN e o SIPE, são planejadores de HTN. Yang (1990)

descreve propriedades de ações que tornam eficiente o planejamento de HTN. Erol, Hendler e Nau (1994, 1996) apresentam um planejador de decomposição hierárquica completo, bem como uma variedade de resultados de complexidade para planejadores de HTN puros.

Nossa apresentação de HTNs e semântica angélica deve-se a Marthi *et al.* (2007, 2008). Kambhampati *et al.* (1998) propuseram uma abordagem em que decomposições são apenas outra forma de refinamento do plano, semelhante aos refinamentos de planejamento de ordem parcial não hierárquica.

A partir do trabalho em macro-operadores em STRIPS, uma das metas do planejamento hierárquico foi a reutilização da experiência de planejamento anterior, sob a forma de planos generalizados. A técnica de **aprendizado baseado na explicação**, descrita em profundidade no Capítulo 19, foi aplicada a vários sistemas como um meio de generalizar planos anteriormente computados, inclusive o SOAR (Laird *et al.*, 1986) e o PRODIGY (Carbonell *et al.*, 1989). Uma abordagem alternativa é armazenar planos anteriormente computados em sua forma original e depois reutilizá-los para resolver por analogia novos problemas semelhantes ao problema original. Essa é a abordagem adotada pelo campo chamado **planejamento baseado em casos** (Carbonell, 1983; Alterman, 1988; Hammond, 1989). Kambhampati (1994) argumenta que o planejamento baseado em casos deve ser analisado como uma forma de planejamento de refinamento e fornece um fundamento formal para o planejamento de ordem parcial baseado em casos.

Os primeiros planejadores, não incluem condicionais e laços, mas podiam usar a coerção para formar planos conformantes. O Noah de Sacerdoti resolveu o problema de “chaves e caixas”, um desafiante problema de planejamento em que o planejador conhece pouco sobre o estado inicial utilizando coerção. Mason (1993) argumenta que, com frequência, o sensoriamento em geral pode e deve ser dispensada no planejamento de robótica, e descreveu um plano sem sensores capaz de mover uma ferramenta para uma posição específica sobre uma mesa por meio de uma sequência de ações, *independentemente* da posição inicial.

Goldman e Boddy (1996) introduziram o termo **conformant planning** (planejamento conformante) para planejadores sem sensores que tratam a incerteza pela coerção do mundo em estados conhecidos, observando que os planos sem sensores frequentemente são efetivos, ainda que o agente tenha sensores. O primeiro planejador conformante moderadamente eficiente foi o Conformant Graphplan ou CGP de Smith e Weld (1998). Ferraris e Giunchiglia (2000) e Rintanen (1999) desenvolveram de forma independente planejadores com conformação baseados no SATPlan. Bonet e Geffner (2000) descrevem um planejador conformante fazendo busca heurística no espaço de estados de crença, com base em ideias iniciais desenvolvidas na década de 1960 para processos decisórios de Markov parcialmente observáveis, ou POMDPs (veja o Capítulo 17).

Atualmente, existem três abordagens principais do planejamento conformante. As duas primeiras utilizam busca heurística no espaço do estado de crença: o HSCP (Bertoli *et al.*, 2001a) utiliza diagramas de decisão binária (BDDs) para representar estados de crença, enquanto Hoffmann e Brafman (2006) adotam a abordagem preguiçosa de cálculo das precondições e dos testes objetivos sobre demanda utilizando solucionadores SAT. A terceira abordagem, defendida principalmente por Jussi Rintanen (2007), formula todo o problema de planejamento sem sensores como uma fórmula booleana quantificada (QBF) e a resolve usando um solucionador QBF de propósito geral. Os planejadores conformantes atuais têm cinco ordens de grandeza mais rápido do que a do CGP. Em

2006, o vencedor da categoria de planejamento conformante na *International Planning Competition* foi o T_0 (Palacios e Geffner, 2007), que usou busca heurística em espaço de estado de crença, enquanto manteve a simples representação do estado de crença pela definição de literais derivados que abrangem efeitos condicionais. Bryce e Kambhampati (2007) discutiram como um grafo de planejamento pode ser generalizado para gerar uma boa heurística para planejamento conformante e contingente.

Tem havido alguma confusão na literatura entre os termos planejamento "condicional" e "contingente". Seguindo Majercik e Littman (2003), usamos "condicional" para nos referir a um plano (ou ação) que tem efeitos diferentes dependendo do estado atual do mundo, e "contingente" para significar um plano no qual o agente pode escolher diferentes ações dependendo dos resultados da detecção. O problema de planejamento contingente recebeu mais atenção após a publicação do artigo influente de Drew McDermott (1978a), *Planning and Acting*.

A abordagem do planejamento contingente descrita nesse capítulo foi baseada em Hoffmann e Brafman (2005), influenciada pelos algoritmos de busca eficientes para grafos cíclicos E-OU desenvolvidos por Jimenez e Torras (2000) e Hansen e Zilberstein (2001). Bertoli *et al.* (2001b) descreveram o MBP (planejamento baseado em modelo), que utiliza diagramas de decisão binária para fazer planejamento conformante e contingente.

Em resumo, agora é possível ver como os principais algoritmos clássicos de planejamento levaram a versões estendidas de domínios incertos. A busca heurística rápida para a frente através do espaço de estados conduziu à busca para a frente no espaço de crença (Bonet e Geffner, 2000; Hoffman e Brafman, 2005); o SATPLAN resultou no SATPLAN estocástico (Majercik e Littman, 2003) e no planejamento com a lógica booleana quantificada (Rintanen, 2007); o planejamento de ordem parcial levou ao UWL (Etzioni *et al*., 1992) e ao CNLP (Peot e Smith, 1992). O GRAPHPLAN conduziu ao Sensory Graphplan ou SGP (Weld *et al*., 1998).

O primeiro planejamento on-line com monitoramento de execução foi o PLANEX (Fikes *et al*., 1972), que funcionava com o planejador STRIPS para controlar o robô Shakey. O planejador NASL (McDermott, 1978a) tratava um problema de planejamento simplesmente como uma especificação para a execução de uma ação complexa, de forma que a execução e o planejamento ficassem completamente unificados.

O SIPE (*System for Interactive Planning and Execution monitoring*) (Wilkins, 1988, 1990) foi o primeiro planejador a lidar sistematicamente com o problema de replanejamento. Ele foi utilizado em projetos de demonstração em vários domínios, inclusive operações de planejamento no convés de voo de um porta-aviões e no escalonamento de linha de produção de uma fábrica da cerveja australiana, e planejando a construção de edifícios com várias lojas (Kartam e Levitt, 1990).

Em meados da década de 1980, o pessimismo sobre os tempos lentos de execução de sistemas de planejamento levou à proposta de agentes reflexos chamados sistemas de **planejamento reativo** (Brooks, 1986; Agre e Chapman, 1987). O PENGI (Agre e Chapman, 1987) foi capaz de jogar um videogame (completamente observável) usando circuitos booleanos combinados a uma representação "visual" de objetivos correntes e do estado interno do agente. Os "planos universais" (Schoppers, 1987, 1989) foram desenvolvidos como um método de busca em tabelas para planejamento reativo, mas acabaram por se tornar uma redescoberta da ideia de **políticas** que foi usada por longo tempo em

processos de decisão de Markov (veja o Capítulo 17). Um plano universal (ou uma política) contém um mapeamento de qualquer estado para a ação que deve ser executada nesse estado. Koenig (2001) pesquisa técnicas de planejamento on-line sob o nome de *Agent-Centered Search*.

O planejamento multiagente teve um salto de popularidade nos últimos anos, embora tenha uma longa história. Konolige (1982) formalizou o planejamento multiagente em lógica de primeira ordem, enquanto Pednault (1986) apresentou uma descrição no estilo de STRIPS. A noção de intenção conjunta, essencial se os agentes tiverem de executar um plano conjunto, vem do trabalho em ações de comunicação (Cohen e Levesque, 1990; Cohen *et al*., 1990). Boutilier e Brafman (2001) mostraram como adaptar o planejamento de ordem parcial para um cenário multiator. Brafman e Domshlak (2008) elaboraram um algoritmo de planejamento multiator cuja complexidade cresce apenas linearmente com o número de atores, desde que o grau de acoplamento (medido em parte, pela **largura da árvore** do grafo de interações entre os agentes) é limitado. Petrik e Zilberstein (2009) mostraram que uma abordagem baseada em programação bilinear supera a abordagem *cover-set* que delineamos nesse capítulo.

Apenas tocamos a superfície do trabalho sobre negociação em planejamento multiagente. Durfee e Lesser (1989) discutem como as tarefas podem ser compartilhadas entre agentes por negociação. Kraus *et al*. (1991) descrevem um sistema para jogar Diplomacy, um jogo de tabuleiro que exige negociação, formação e dissolução de alianças e desonestidade. Stone (2000) mostra como agentes podem cooperar como participantes de equipes no ambiente competitivo, dinâmico e parcialmente observável do futebol de robôs. Em um artigo posterior, Stone (2003) analisa dois ambientes multiagentes competitivo — RoboCup, uma competição de futebol de robôs, e TAC, o Trading Agents Competition baseado em leilão —, e descobre que a intratabilidade computacional de nossas abordagens atuais, teoricamente bem fundamentadas, fez com que muitos sistemas multiagentes fossem projetados por métodos *ad hoc*.

Em sua teoria altamente influente, *Society of Mind*, Marvin Minsky (1986, 2007) expôs que as mentes humanas são construídas de um conjunto de agentes. Livnat e Pippenger (2006) provaram que, para o problema de encontrar o melhor caminho, e dada uma limitação da quantidade total dos recursos de computação, a melhor arquitetura para um agente é um conjunto de subagentes, cada um dos quais tenta otimizar seu próprio objetivo, e todos estão em conflito uns com os outros.

O modelo *boid* se deve a Reynolds (1987), que ganhou um prêmio da academia por sua aplicação a bandos de morcegos e de pinguins no filme *Batman, o retorno*. O jogo NERO e os métodos de estratégias de aprendizagem são descritos por Bryant e Miikkulainen (2007).

Livros recentes sobre sistemas multiagentes incluem os de Weiss (2000a), Young (2004), Vlassis (2008) e Shoham e Leyton-Brown (2009). Existe uma conferência anual sobre agentes autônomos e sistemas multiagentes (AAMAS).

EXERCÍCIOS

11.1 Todos os objetivos que consideramos até aqui solicitam ao planejador que o mundo satisfaça o objetivo em apenas um passo. Nem todos os objetivos podem ser expressos dessa forma: não se atinge o objetivo de suspender um lustre jogando-o no ar. Falando mais sério, ninguém desejaria que

o sistema de apoio à vida de uma nave fornecesse oxigênio em um dia e em outro não. O *objetivo de manutenção* é atingido quando o plano do agente causa uma condição verdadeira contínua de um determinado estado em diante. Descreva como estender o formalismo desse capítulo para tratar objetivos de manutenção.

11.2 Você tem uma quantidade de caminhões com os quais quer entregar um conjunto de pacotes. Cada pacote começa em algum local em uma grade e tem um destino em outro lugar. Cada caminhão é controlado diretamente para mover-se para a frente e virar. Construa uma hierarquia de ações de alto nível para esse problema. Que conhecimento sobre a solução a sua hierarquia codifica?

11.3 Suponha que uma ação de alto nível tenha exatamente uma implementação como uma sequência de ações primitivas. Dê um algoritmo para calcular as suas precondições e efeitos, dada a hierarquia completa de refinamento e os esquemas das ações primitivas.

11.4 Suponha que o conjunto otimista alcançável de um plano de alto nível seja um superconjunto do conjunto objetivo; algo pode ser concluído sobre se o plano atinge o objetivo? E se o conjunto pessimista alcançável não tiver interseção com o conjunto objetivo? Explique.

11.5 Escreva um algoritmo que tome um estado inicial (especificado por um conjunto de literais proposicionais) e uma sequência de HLAs (cada um definido por precondições e especificações angélicas de conjuntos alcançáveis otimistas e pessimistas) e compute as descrições otimista e pessimista do conjunto alcançável da sequência.

11.6 Na Figura 11.2, mostramos como descrever ações em um problema de escalonamento usando campos separados para DURAÇÃO, USO e CONSUMO. Agora, suponha que desejássemos combinar o escalonamento com o planejamento não determinístico, que exige efeitos não determinísticos e condicionais. Considere cada um dos três campos e explique se eles devem permanecer separados ou se devem tornar-se efeitos da ação. Dê um exemplo para cada um dos três.

11.7 Algumas operações em linguagens de programação-padrão podem ser modeladas como ações que alteram o estado do mundo. Por exemplo, a operação de atribuição altera o conteúdo de uma posição de memória, enquanto a operação de impressão muda o estado do fluxo de saída. Um programa que consiste nessas operações também pode ser considerado um plano cujo objetivo é dado pela especificação do programa. Por conseguinte, algoritmos de planejamento podem ser usados para construir programas que alcançam uma determinada especificação.

- **a.** Escreva um esquema de ação para o operador de atribuição (atribuindo o valor de uma variável a outra). Lembre-se de que o valor original será sobrescrito!
- **b.** Mostre como a criação de objetos pode ser utilizada por um planejador para produzir um plano que vai trocar os valores de duas variáveis usando uma variável temporária.

11.8 Suponha que a ação *Troca* sempre altere o valor verdadeiro de *L*. Mostre como definir os seus efeitos usando um esquema de ação com efeitos condicionais. Mostre que, apesar do uso de efeitos condicionais, a representação do estado de crença 1-FNC permanece em 1-FNC após *Troca*.

11.9 No mundo de blocos, fomos forçados a introduzir dois esquemas de ações de STRIPS, *Mover* e *MoverParaMesa*, a fim de manter o predicado *Livre* de forma correta. Mostre como os efeitos condicionais podem ser usados para representar ambos os casos com uma única ação.

11.10 Os efeitos condicionais foram ilustrados para a ação *Aspirar* no mundo de aspirador de pó — o quadrado que fica limpo depende do quadrado em que o robô se encontra. Você poderia imaginar um novo conjunto de variáveis proposicionais para definir estados do mundo de aspirador de pó, tais que *Aspirar* tenha uma descrição *incondicional*? Represente as descrições de *Aspirar*, *Esquerda* e *Direita* usando suas proposições e demonstre que elas bastam para descrever todos os estados possíveis do mundo.

11.11 Encontre um tapete apropriadamente sujo, livre de obstáculos, e limpe-o com o aspirador. Desenhe o caminho feito pelo aspirador o mais preciso que puder. Explique o resultado fazendo referência às formas de planejamento discutidas neste capítulo.

11.12 Adicione ao problema do medicamento um *Teste* de ação que tenha o efeito condicional *CrescimentoCultura* quando *Doença* for verdadeira e que em qualquer caso tenha o efeito perceptivo *Conhecido*(*CrescimentoCultura)*. Esboce um plano condicional que resolva o problema e minimize o uso da ação *Medicar*.

---

[1] Planejadores HTN muitas vezes permitem refinamento em planos parcialmente ordenados, e permitem refinamentos de duas HLAs diferentes em um plano para *compartilhar* ações. Omitimos essas complicações importantes no interesse do entendimento dos conceitos básicos de planejamento hierárquico.

[2] Se, para um problema não determinístico, são necessárias soluções cíclicas, a busca E-OU deve ser generalizada para uma versão de laço como LAO* (Hansen e Zilberstein, 2001).

[3] Em 1954, a Sra. Hodges do Alabama foi atingida por um meteorito que caiu através de seu telhado. Em 1992, um pedaço do meteorito Mbale atingiu um menino na cabeça; felizmente, sua descida foi amortecida por folhas de bananeira (Jenniskens *et al.*, 1994). Em 2009, um menino alemão alegou ter sido atingido na mão por um meteorito do tamanho de uma ervilha. Esses incidentes não resultaram em nenhuma lesão grave, sugerindo que a necessidade de pré-planejamento contra tais contingências às vezes é exagerada.

[4] Monitoramento de plano significa que finalmente, depois de 371 páginas, temos um agente que é mais esperto do que um besouro de esterco. Um agente de monitoramento de plano notaria que uma bola de esterco estava faltando do seu alcance e iria replanejar para obter outra bola e tampar o buraco.

[5] Repetição inútil do reparo de um plano é exatamente o comportamento exibido pela vespa Sphex.

[6] Pedimos desculpas aos residentes do Reino Unido, onde o simples ato de contemplar uma partida de tênis é garantia de chuva.

CAPÍTULO 12

Representação de conhecimento

Em que mostramos como usar a lógica de primeira ordem para representar os aspectos mais importantes do mundo real, como ação, espaço, tempo, pensamentos e compras.

Os capítulos anteriores descreveram a tecnologia dos agentes baseados em conhecimento: a sintaxe, a semântica e a teoria de prova da lógica proposicional e da lógica de primeira ordem, e ainda a implementação de agentes que utilizam essas lógicas. Neste capítulo, examinaremos a questão do *conteúdo* que deve ser colocado na base de conhecimento de tal agente, ou seja, como representar fatos sobre o mundo.

A Seção 12.1 introduz a ideia de uma ontologia geral, que organiza tudo no mundo em uma hierarquia de categorias. A Seção 12.2 abrange as categorias de objetos, substâncias e medidas; a Seção 12.3 abrange eventos, e a Seção 12.4 discute o conhecimento sobre crenças. Então voltamos a considerar a tecnologia para raciocínio com este conteúdo: a Seção 12.5 discute sistemas de raciocínio projetados para inferência eficiente com categorias e a Seção 12.6 discute o raciocínio com informação default. A Seção 12.7 junta todo o conhecimento no contexto de um ambiente de compra pela Internet.

12.1 ENGENHARIA ONTOLÓGICA

Em domínios em "miniatura", a escolha da representação não é tão importante; muitas escolhas vão funcionar. Por outro lado, domínios complexos como compras na Internet ou dirigir um carro no trânsito exigem representações mais gerais e flexíveis. Este capítulo mostra como criar essas representações, concentrando-se em conceitos gerais — como *Eventos*, *Tempo*, *Objetos Físicos* e *Crenças* — que ocorrem em muitos domínios diferentes. A representação desses conceitos abstratos é chamada às vezes de **engenharia ontológica**.

A perspectiva de representar *tudo* no mundo é assustadora. É claro que não escreveremos realmente uma descrição completa de tudo — isso seria demais até mesmo para um livro didático de 1.000 páginas —, mas deixaremos espaços vazios onde poderá ser colocado conhecimento sobre qualquer domínio. Por exemplo, definiremos o que significa ser um objeto físico, mas os detalhes de diferentes tipos de objetos — robôs, televisores, livros ou qualquer outro objeto — poderão ser

preenchidos mais tarde. Isso é análogo à forma como os projetistas de uma estrutura de programação orientada a objetos (como a estrutura gráfica Java Swing) definem conceitos gerais como *Janela*, esperando que os usuários o utilizem para definir conceitos mais específicos como *SpreadsheetWindow*. A estrutura geral de conceitos é chamada **ontologia superior**, devido à convenção de desenhar grafos com os conceitos gerais na parte superior e os conceitos mais específicos abaixo deles, como na Figura 12.1.

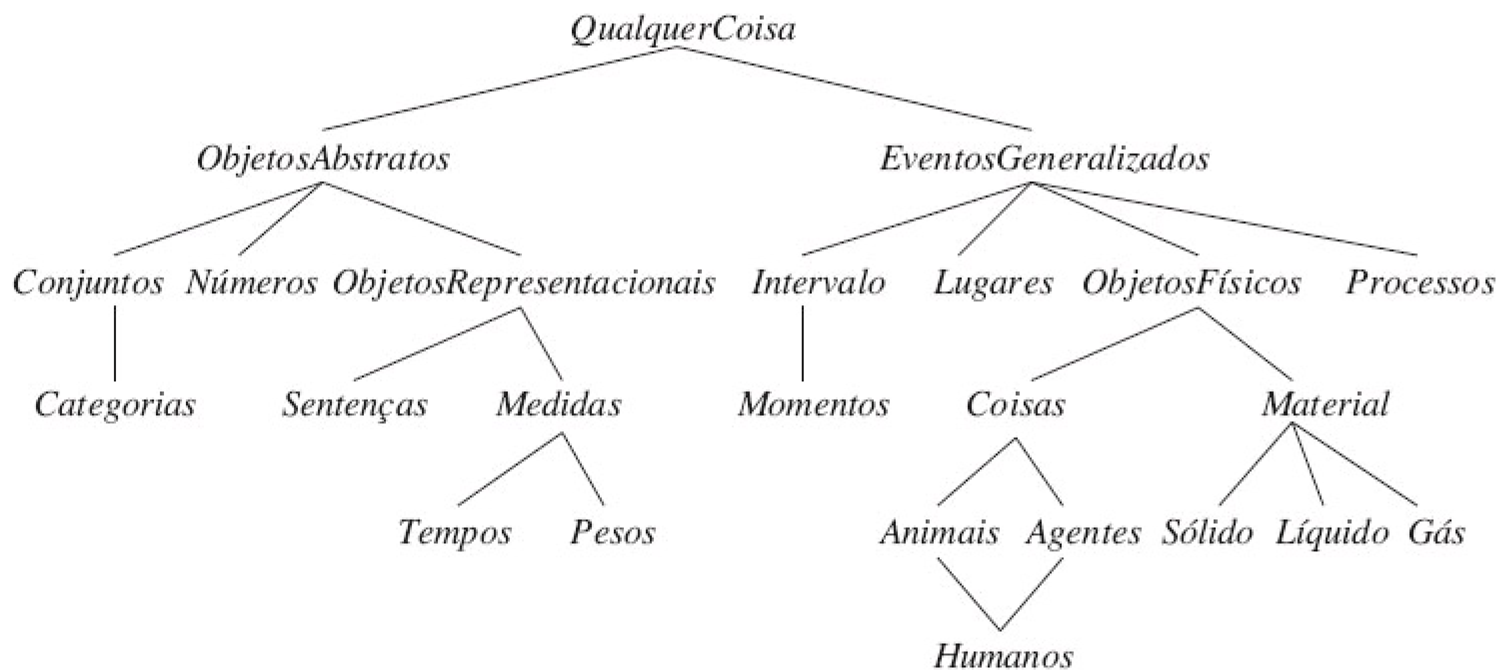


Figura 12.1 A ontologia superior do mundo, mostrando os tópicos a serem cobertos mais adiante neste capítulo. Cada aresta indica que o conceito inferior é uma especialização do conceito superior. As especializações não são necessariamente disjuntas: um ser humano é animal e agente, por exemplo. Veremos, na Seção 12.3.3, por que os objetos físicos aparecem sob eventos generalizados.

Antes de considerarmos a ontologia com mais detalhes, devemos fazer uma importante advertência. Optamos por utilizar a lógica de primeira ordem para discutir o conteúdo e a organização do conhecimento, apesar de que certos aspectos do mundo real são difíceis de captar em LPO. A principal dificuldade é que a maioria das generalizações têm exceções ou só são válidas até certo ponto. Por exemplo, embora "tomates são vermelhos" seja uma regra útil, alguns tomates são verdes, amarelos ou alaranjados. Exceções semelhantes podem ser encontradas em quase todas as regras gerais deste capítulo. A habilidade de tratar exceções e a incerteza é extremamente importante, mas é ortogonal à tarefa de compreender a ontologia geral. Por essa razão, adiaremos a discussão de exceções até a Seção 12.5 deste capítulo e, a do tópico mais geral de raciocínio com incerteza, até o Capítulo 13.

Qual é a utilidade de uma ontologia superior? Considere novamente a ontologia dos circuitos da Seção 8.4.2. Ela faz muitas suposições simplificadoras. Por exemplo, o tempo é completamente omitido, sinais são fixos e não de propagam, a estrutura do circuito permanece constante. Uma ontologia mais geral consideraria sinais em instantes particulares e incluiria os comprimentos dos fios e retardos de propagação. Isso nos permitiria simular as propriedades de sincronização do circuito e, na verdade, tais simulações são executadas com frequência por projetistas de circuitos. Também poderíamos introduzir classes mais interessantes de portas, descrevendo por exemplo a

tecnologia (TTL, CMOS, e assim por diante), bem como a especificação de entrada/saída. Se quiséssemos discutir a confiabilidade ou o diagnóstico, incluiríamos a possibilidade de a estrutura do circuito ou as propriedades das portas poderem se alterar espontaneamente. Para levar em conta capacitâncias espúrias, precisaríamos representar onde os fios estão na placa.

Se observarmos o mundo de wumpus, veremos que se aplicam considerações semelhantes. Embora seja incluído o tempo, ele tem uma estrutura muito simples: nada acontece exceto quando o agente age, e todas as mudanças são instantâneas. Uma ontologia mais geral, mais bem adaptada ao mundo real, permitiria que mudanças simultâneas se estendessem ao longo do tempo. Também usamos um predicado *Poço* para dizer que quadrados têm poços. Poderíamos ter permitido tipos diferentes de poços, fazendo com que diversos indivíduos pertencessem à classe de poços, cada um com propriedades diferentes. De modo semelhante, poderíamos permitir outros animais além de wumpus. Não seria possível definir as espécies exatas das percepções disponíveis e, assim, precisaríamos construir uma taxonomia biológica do mundo de wumpus para ajudar o agente a prever o comportamento de moradores de caverna a partir de pistas escassas.

Para qualquer ontologia de uso específico, é possível fazer mudanças como essas com a finalidade de obter uma generalidade maior. Uma questão óbvia surge então: todas essas ontologias convergem para uma ontologia de uso geral? Depois de séculos de investigação filosófica e computacional, a resposta é "talvez". Nesta seção apresentaremos uma ontologia de propósito geral que sintetiza as ideias daqueles séculos. Existem duas características importantes das ontologias de uso geral que as distinguem das coleções de ontologias de uso específico:

- Uma ontologia de uso geral deve ser aplicável em quase todo domínio de uso específico (com a inclusão de axiomas específicos do domínio). Isso significa que, nenhuma questão de representação pode ser tratada com artimanhas nem pode ser empurrada para debaixo do tapete.
- Em qualquer domínio suficientemente exigente, áreas distintas de conhecimento devem ser *unificadas* porque o raciocínio e a resolução de problemas poderiam envolver diversas áreas simultaneamente. Por exemplo, um sistema robô para reparação de circuitos precisa raciocinar sobre circuitos em termos de conectividade elétrica e leiaute físico, e também sobre o tempo, tanto para realizar a análise de sincronização de circuitos quanto para avaliar os custos do trabalho. Portanto, as sentenças que descrevem o tempo devem ser capazes de se combinar com as que descrevem o leiaute espacial e devem funcionar igualmente bem para nanossegundos e minutos, e para angstrons e metros.

Deveríamos dizer de antemão que o empreendimento da engenharia ontológica geral tem tido até agora sucesso limitado. Nenhum dos aplicativos *top* de IA (conforme listado no Capítulo 1) utiliza uma ontologia compartilhada — todos utilizam a engenharia do conhecimento para fins específicos. As considerações sociais/políticas podem tornar difícil para os grupos concorrentes chegar a um acordo sobre uma ontologia. Como Tom Gruber (2004) diz: "Cada ontologia é um tratado — um acordo social — entre pessoas que têm um motivo comum para compartilhar." Quando o interesse da concorrência supera a motivação para compartilhar, não pode haver ontologia comum. As ontologias que já existem foram criadas ao longo de quatro rotas:

1. Por uma equipe de ontologistas/lógicos treinados, que arquitetaram a ontologia e escreveram

axiomas. O sistema CYC foi quase todo construído dessa forma (Lenat e Guha, 1990).

2. Através da importação de categorias, atributos e valores de um banco de dados existente ou bancos de dados. O DBPedia foi construído através da importação de fatos estruturados da Wikipedia (Bizer *et al*., 2007).
3. Ao analisar os documentos de texto e extrair informações deles. O TextRunner foi construído pela leitura de um grande *corpus* de páginas da Web (Banko e Etzioni, 2008).
4. Ao estimular os amadores não qualificados para fornecer conhecimento do senso comum. O sistema OPENMIND foi construído por voluntários que propuseram fatos em inglês (Singh *et al*., 2002.; Chklovski e Gil, 2005).

12.2 CATEGORIAS E OBJETOS

A organização de objetos em **categorias** é uma parte vital da representação de conhecimento. Embora a interação com o mundo ocorra no nível de objetos individuais, *uma grande parte do raciocínio ocorre no nível de categorias*. Por exemplo, um consumidor teria normalmente o objetivo de comprar uma bola de basquete, e não *determinada* bola de basquete, como a BB_9. As categorias também servem para fazer previsões sobre objetos, uma vez que eles estão classificados. Pode-se deduzir a presença de certos objetos a partir de percepções, deduzir a pertinência a uma categoria a partir das propriedades percebidas dos objetos e depois utilizar as informações a respeito da categoria para fazer previsões sobre os objetos. Por exemplo, a partir de seu grande tamanho, da casca verde e amarela e da forma ovoide, carne vermelha, sementes pretas e a presença no corredor de frutas, pode-se deduzir que um objeto é uma melancia; a partir disso, é possível deduzir que seria útil em uma salada de frutas.

Há duas opções para representar categorias em lógica de primeira ordem: predicados e objetos. Isto é, podemos usar o predicado *BolaDeBasquete*(*b*) ou podemos **reificar**[1] a categoria sob a forma de um objeto *BolasDeBasquete*. Então, poderíamos afirmar: *Elemento*(*b*, *BolasDeBasquete*) (que abreviaremos como $b \in BolasDeBasquete$) para dizer que *b* é um elemento da categoria de bolas de basquete. Afirmamos: *Subconjunto*(*BolasDeBasquete*, *Bolas*), abreviado como $BolasDeBasquete \subset Bolas$, para dizer que *BolasDeBasquete* é uma **subcategoria** de *Bolas*. Utilizaremos subcategoria, subclasse e subconjunto indistintamente.

As categorias servem para organizar e simplificar a base de conhecimento por **herança**. Se dissermos que todas as instâncias da categoria *Alimento* são comestíveis e se afirmarmos que *Fruta* é uma subclasse de *Alimento* e que *Maçãs* é uma subclasse de *Fruta*, então saberemos que toda maçã é comestível. Dizemos que as maçãs individuais **herdam** a propriedade de serem comestíveis, nesse caso de sua condição de elemento da categoria *Alimento*.

As relações de subclasses organizam as categorias em uma **taxonomia** ou **hierarquia taxonômica**. As taxonomias foram usadas explicitamente durante séculos em campos técnicos. A maior destas taxonomias organiza cerca de 10 milhões de espécies vivas e extintas, muitas delas besouros,[2] em uma única hierarquia; a biblioteconomia desenvolveu uma taxonomia de todos os campos do conhecimento, codificada como sistema decimal de Dewey; e as autoridades tributárias e outros

departamentos governamentais desenvolveram taxonomias extensas de ocupações e produtos comerciais. As taxonomias também constituem um aspecto importante do conhecimento comum em geral.

A lógica de primeira ordem facilita a declaração de fatos sobre categorias, seja relacionando objetos a categorias ou pela quantificação de seus elementos. Apresentamos alguns tipos de fatos, com exemplos:

- Um objeto é um elemento de uma categoria.
  $BB_9 \in BolasDeBasquete$
- Uma categoria é uma subclasse de outra categoria.
  $BolasDeBasquete \subset Bolas$
- Todos os elementos de uma categoria têm algumas propriedades.
  $(x \in BolasDeBasquete) \Rightarrow Esférica(x)$
- Os elementos de uma categoria podem ser reconhecidos por algumas propriedades.
  $Laranja(x) \land Redonda(x) \land Diâmetro(x) = 23{,}75 \text{ cm} \land x \in Bolas \Rightarrow x \in BolasDeBasquete$
- Uma categoria é um conjunto que tem algumas propriedades.
  $Cães \in EspéciesDomesticadas$

Note que, pelo fato de *Cães* ser uma categoria e ser um elemento de *EspéciesDomesticadas*, esta última deve ser uma categoria de categorias.

Claro que há exceções a muitas dessas regras (a bola de basquete furada não é esférica); lidaremos com essas exceções mais tarde.

Embora as relações de subclasse e elemento sejam as mais importantes para categorias, também queremos ter a possibilidade de enunciar relações entre categorias que não são subclasses umas das outras. Por exemplo, se afirmássemos simplesmente que *Machos* e *Fêmeas* são subclasses de *Animais*, não teríamos dito que o macho não pode ser uma fêmea. Dizemos que duas ou mais categorias são **disjuntas** se elas não têm elementos (ou membros) em comum. Além disso, mesmo se soubermos que machos e fêmeas são disjuntos, não saberemos que um animal que não é um macho tem de ser uma fêmea, a menos que digamos que machos e fêmeas constituem uma **decomposição exaustiva** dos animais. Uma decomposição exaustiva desjunta é chamada **partição**. Os exemplos a seguir ilustram esses três conceitos:

$Disjuntos(\{Animais, Vegetais\})$
$DecomposiçãoExaustiva(\{Americanos, Canadenses, Mexicanos\}, NorteAmericanos)$
$Partição(\{Machos, Fêmeas\}, Animais).$

(Observe que a *DecomposiçãoExaustiva* de *NorteAmericanos* não é uma *Partição* porque algumas pessoas têm dupla cidadania.) Os três predicados são definidos como:

$Disjuntos(s) \Leftrightarrow (\forall c_1, c_2 \; c_1 \in s \land c_2 \in s \land c_1 \neq c_2 \Rightarrow Interseção(c_1, c_2) = \{\})$
$DecomposiçãoExaustiva(s, c) \Leftrightarrow (\forall i \; i \in c \Leftrightarrow \exists c_2 \; c_2 \in s \land i \in c_2)$
$Partição(s, c) \Leftrightarrow Disjuntos(s) \land DecomposiçãoExaustiva(s, c).$

As categorias também podem ser *definidas* fornecendo-se condições necessárias e suficientes para pertinência. Por exemplo, um solteiro é um macho adulto não casado:

$$x \in Solteiros \Leftrightarrow NãoCasado(x) \land x \in Adultos \land x \in Machos.$$

Como discutimos no quadro sobre espécies naturais mais adiante, as definições lógicas estritas para categorias nem sempre são possíveis e nem sempre são necessárias.

12.2.1 Composição física

A ideia de que um objeto pode fazer parte de outro é bastante familiar. O nariz de uma pessoa faz parte da cabeça de uma pessoa, a Romênia faz parte da Europa, e este capítulo faz parte deste livro. Usamos a relação geral *ParteDe* para dizer que alguma coisa faz parte de outra. Os objetos podem ser agrupados em hierarquias de *ParteDe*, uma reminiscência da hierarquia de *Subconjunto*:

$ParteDe(Bucareste, Romênia)$
$ParteDe(Romênia, EuropaOriental)$
$ParteDe(EuropaOriental, Europa)$
$ParteDe(Europa, Terra).$

A relação *ParteDe* é transitiva e reflexiva, ou seja:

$ParteDe(x, y) \land ParteDe(y, z) \Rightarrow ParteDe(x, z).$
$ParteDe(x, x).$

Portanto, podemos concluir: $ParteDe(Bucareste, Terra)$.

As categorias de **objetos compostos** frequentemente são caracterizadas por relações estruturais entre partes. Por exemplo, um bípede tem duas pernas presas a um corpo:

$$\begin{aligned} Bípede(a) \;\Rightarrow\; & \exists l_1, l_2, b \; Perna(l_1) \land Perna(l_2) \land Corpo(b) \land \\ & ParteDe(l_1, a) \land ParteDe(l_2, a) \land ParteDe(b, a) \land \\ & Presa(l_1, b) \land Presa(l_2, b) \land \\ & l_1 \neq l_2 \land [\forall l_3 \; Perna(l_3) \land ParteDe(l_3, a) \Rightarrow (l_3 = l_1 \lor l_3 = l_2)]. \end{aligned}$$

A notação correspondente a “exatamente dois” é um pouco confusa; somos forçados a afirmar que existem duas pernas, que elas não são iguais e que, se alguém propuser uma terceira perna, ela terá de ser igual a uma das outras duas. Na Seção 12.5.2, veremos que um formalismo chamado *lógica de descrições* facilita a representação de restrições como “exatamente dois”.

Podemos definir uma relação *PartiçãoDeParte* análoga à relação *Partição* para categorias (veja o

Exercício 12.8). Um objeto é composto das partes de sua *PartiçãoDeParte* e pode ser visualizado como derivando algumas propriedades dessas partes. Por exemplo, a massa de um objeto composto é a soma das massas das partes. Note que isso não ocorre no caso de categorias, que não têm massa, embora seus elementos possam ter.

Também é útil definir objetos compostos com partes definidas mas sem estrutura específica. Por exemplo, poderíamos dizer: "As maçãs deste pacote pesam 900 gramas." A tentação seria atribuir esse peso ao *conjunto* de maçãs do pacote, mas isso seria um erro porque o conjunto é um conceito matemático abstrato que tem elementos, mas não tem peso. Em vez disso, precisamos de um novo conceito, que chamaremos de **grupo**. Por exemplo, se as maçãs são $Maçã_1$, $Maçã_2$ e $Maçã_3$, então:

$$GrupoDe(\{Maçã_1, Maçã_2, Maçã_3\})$$

denota o objeto composto com as três maçãs como partes (não como elementos). Podemos então usar o grupo como um objeto normal, embora não estruturado. Note que $GrupoDe(\{x\}) = x$. Além disso, *GrupoDe*(*Maçãs*) é o objeto composto que consiste em todas as maçãs — não devemos confundi-lo com *Maçãs*, a categoria ou o conjunto de todas as maçãs.

ESPÉCIES NATURAIS

Algumas categorias têm definições estritas: um objeto é um triângulo se e somente se é um polígono com três lados. Por outro lado, a maioria das categorias no mundo real não tem nenhuma definição clara; essas são as chamadas categorias de **espécies naturais**. Por exemplo, tomates tendem a ser vermelhos, aproximadamente esféricos, com uma depressão na parte superior em que fica o talo, têm cerca de 5-10 cm de diâmetro, apresentam pele fina mas resistente e têm polpa, sementes e suco em seu interior. Porém, há variações: alguns tomates são amarelos, tomates não maduros são verdes, alguns são menores ou maiores que a média, e tomates cereja são uniformemente pequenos. Em vez de uma definição completa de tomates, temos um conjunto de características que serve para identificar objetos que são claramente tomates típicos, mas que não permite decidir para outros objetos (é possível existir um tomate com a casca aveludada como um pêssego?).

Isso representa um problema para um agente lógico. O agente não pode ter certeza de que um objeto que ele percebeu é um tomate e, mesmo que tivesse certeza disso, poderia não estar certo de quais das propriedades de tomates típicos esse objeto apresenta. Esse problema é uma consequência inevitável da operação em ambientes parcialmente observáveis.

Uma abordagem útil é separar o que é verdadeiro para todas as instâncias de uma categoria daquilo que é verdadeiro apenas para instâncias típicas. Assim, além da categoria *Tomates*, também teremos a categoria *Típica*(*Tomates*). Aqui, a função *Típica* mapeia uma categoria como a subclasse que contém apenas as instâncias típicas:

$$Típica\ (c) \subseteq c.$$

A maior parte do conhecimento sobre tipos naturais será na realidade o conhecimento sobre

suas instâncias típicas:

$x \in$ *Típica (Tomates)* $\Rightarrow$ *Vermelho (x)* $\wedge$ *Redondo (x)*.

Desse modo, podemos anotar fatos úteis sobre categorias sem definições exatas. A dificuldade de fornecer definições exatas para a maioria das categorias naturais foi explicada em profundidade por Wittgenstein (1953). Ele utilizou o exemplo de *jogos* para mostrar que elementos de uma categoria compartilham "semelhanças familiares", em vez de características necessárias e suficientes. Que definição estrita engloba xadrez, pega-pega, paciência e queimada?

A utilidade da noção de definição estrita também foi contestada por Quine (1953). Ele destacou que até mesmo a definição "solteiro" como um macho adulto não casado é suspeita; por exemplo, alguém poderia questionar uma declaração como "o papa é solteiro". Embora não seja estritamente falso, esse uso é sem dúvida pouco feliz porque induz inferências involuntárias por parte do ouvinte. A tensão talvez pudesse ser resolvida distinguindo entre definições lógicas apropriadas para representação interna do conhecimento e os critérios mais ricos em nuanças do uso linguístico adequado. Este último pode ser alcançado "filtrando-se" as asserções derivadas do primeiro. Também é possível que as falhas do uso linguístico sirvam como *feedback* para a modificação de definições internas, de modo que a filtragem se torne desnecessária.

Podemos definir *GrupoDe* em termos da relação *ParteDe*. É óbvio que cada elemento de *s* é parte de *Grupo-De*(*s*):

$\forall x\ x \in s \Rightarrow ParteDe(x, GrupoDe(\text{s}))$.

Além disso, *GrupoDe*(*s*) *é o menor objeto que satisfaz essa condição*. Em outras palavras, *GrupoDe*(*s*) deve fazer parte de qualquer objeto que tenha todos os elementos de *s* como partes:

$\forall y\ [\forall x\ x \in s \Rightarrow ParteDe(x, y)] \Rightarrow ParteDe(GrupoDe(s), y)$.

Esses axiomas são um exemplo de uma técnica geral chamada **minimização lógica**, que significa definir um objeto como o menor que satisfaz certas condições.

12.2.2 Medições

Tanto na teoria científica quanto na teoria de senso comum do mundo, os objetos têm altura, massa, custo, e assim por diante. Os valores que atribuímos para essas propriedades são chamados **medidas**. As medidas quantitativas comuns são bastante fáceis de representar. Imaginamos que o universo inclui "objetos de medida" abstratos, como o *comprimento*, que é o comprimento deste segmento de linha: |————————|. Podemos chamar esse comprimento de 1,5 polegada ou 3,81 centímetros. Desse modo, o mesmo comprimento tem diferentes nomes em nossa linguagem. Representamos o comprimento com uma **função de unidade** que recebe um número como argumento (um esquema alternativo é explorado no Exercício 12.9). Se o segmento de linha for chamado de L_1, poderemos

escrever:

$$Comprimento(L_1) = Polegadas(1{,}5) = Centímetros(3{,}81).$$

A conversão entre unidades é feita igualando-se os múltiplos de uma unidade aos de outra:

$$Centímetros(2{,}54 \times d) = Polegadas(d).$$

Axiomas semelhantes podem ser escritos para libras e quilogramas, segundos e dias, e ainda dólares e centavos. As medidas podem ser usadas para descrever objetos como:

$$Diâmetro(BolaDeBasquete_{12}) = Centímetros(23{,}75).$$
$$Preço(BolaDeBasquete_{12}) = \$(19).$$
$$d \in Dias \Rightarrow Duração(d) = Horas(24).$$

Observe que $(1) *não* é uma nota de um dólar! Podemos ter duas notas de um dólar, mas só existe um objeto denominado $(1). Observe também que, embora *Polegadas*(0) e *Centímetros*(0) se refiram ao mesmo comprimento zero, eles não são idênticos a outras medidas de zero, como *Segundos*(0).

Medidas simples e quantitativas são fáceis de representar. Outras medidas representam um problema maior porque não têm nenhuma escala de valores estabelecida. Os exercícios têm dificuldade, sobremesas são deliciosas e os poemas têm beleza, embora não se possa atribuir aos números essas qualidades. Em um momento de puro caráter contábil, alguém poderia desprezar tais propriedades como inúteis para o propósito do raciocínio lógico ou, pior ainda, tentar impor uma escala numérica sobre a beleza. Isso seria um equívoco grave porque é desnecessário. O aspecto mais importante das medidas não é o dos valores numéricos específicos, mas o fato de que as medidas podem ser *ordenadas*.

Embora as medidas não sejam números, ainda podemos compará-las usando um símbolo de ordenação como >. Por exemplo, poderíamos muito bem acreditar que os exercícios de Norvig são mais trabalhosos que os de Russell e isso daria uma pontuação menor aos exercícios mais trabalhosos:

$$e_1 \in Exercícios \land e_2 \in Exercícios \land Escreveu(Norvig, e_1) \land Escreveu(Russell, e_2) \Rightarrow$$
$$Dificuldade(e_1) > Dificuldade(e_2).$$
$$e_1 \in Exercícios \land e_2 \in Exercícios \land Dificuldade(e_1) > Dificuldade(e_2) \Rightarrow$$
$$PontuaçãoEsperada(e_1) < PontuaçãoEsperada(e_2).$$

Isso é suficiente para permitir que alguém decida que exercícios fazer, embora nenhum valor numérico referente à dificuldade tenha sido usado (porém, alguém teria de descobrir quem escreveu cada um dos exercícios).

Esses tipos de relacionamentos monotônicos entre medidas formam a base para o campo da **física qualitativa**, um subcampo da IA que investiga como raciocinar sobre sistemas físicos sem mergulhar em equações detalhadas e simulações numéricas. A física qualitativa é discutida na seção de notas

históricas.

12.2.3 Objetos: materiais e coisas

O mundo real pode ser visto como a reunião de objetos primitivos (por exemplo, partículas atômicas) e objetos compostos construídos a partir deles. Raciocinando no nível de grandes objetos como maçãs e carros, podemos superar a complexidade envolvida no trato individual com imenso número de objetos primitivos. No entanto, existe uma parte significativa da realidade que parece desafiar qualquer **individuação** óbvia — a divisão em objetos distintos. Daremos a essa parte o nome genérico de **material**. Por exemplo, vamos supor que eu tenha um pouco de manteiga e um tamanduá na minha frente. Posso dizer que existe um tamanduá, mas não existe nenhum número óbvio de "objetos manteiga" porque qualquer parte de um objeto manteiga também é um objeto manteiga, pelo menos até chegarmos a partes muito pequenas. Essa é a principal distinção entre o material e os objetos. Se cortarmos um tamanduá ao meio, não obteremos dois tamanduás (infelizmente).

A linguagem natural faz uma clara distinção entre *material e coisas*. Dizemos "um tamanduá", mas, exceto em restaurantes da pretensiosa Califórnia, não se pode dizer "uma manteiga". Os linguistas fazem distinção entre **substantivos contáveis**, como tamanduás, poços e teoremas, e **substantivos de massa**, como manteiga, água e energia. Várias ontologias concorrentes afirmam poder tratar essa distinção. Descreveremos apenas uma; as outras serão abordadas na seção de notas históricas.

Para representar corretamente um *material*, começamos com o óbvio. Precisaremos ter como objetos em nossa ontologia pelo menos as "massas" brutas de *material* com que interagimos. Por exemplo, poderíamos reconhecer uma massa de manteiga como a mesma manteiga que foi deixada sobre a mesa na noite anterior; poderíamos pegá-la, pesá-la e vendê-la ou ainda realizar qualquer outra ação. Nesse sentido, ela é um objeto semelhante ao tamanduá. Vamos chamá-lo $Manteiga_3$. Também definiremos a categoria *Manteiga*. Informalmente, seus elementos serão itens sobre os quais se poderia dizer "é manteiga" inclusive $Manteiga_3$. Com algumas advertências sobre partes muito pequenas que omitiremos por enquanto, qualquer parte de um objeto manteiga também é um objeto manteiga:

$$b \in Manteiga \land ParteDe(p, b) \Rightarrow p \in Manteiga.$$

Agora, podemos dizer que a manteiga derrete a aproximadamente 30 graus Celsius:

$$b \in Manteiga \Rightarrow PontoDeFusão(b, Celsius(30)).$$

Poderíamos continuar a dizer que a manteiga é amarela, menos densa que a água, é mole à temperatura ambiente, tem alto conteúdo de gordura, e assim por diante. Por outro lado, a manteiga não tem nenhum tamanho específico nem forma, nem peso. Podemos definir categorias mais especializadas de manteiga como *ManteigaSemSal*, que também é uma espécie de *material*. Por outro lado, a categoria *PacoteDeManteiga*, que inclui como elementos todos os objetos manteiga com o peso de 250 gramas, não é uma espécie de *material*. Se cortarmos um pacote de manteiga ao meio, infelizmente não teremos dois pacotes de manteiga.

Na realidade, o que temos é que existem algumas propriedades que são **intrínsecas**: elas são pertinentes à substância do objeto, e não ao objeto como um todo. Quando se corta uma instância do *material* ao meio, as duas partes retêm o mesmo conjunto de propriedades intrínsecas — itens como densidade, ponto de ebulição, sabor, cor, e assim por diante. Por outro lado, propriedades **extrínsecas** — peso, sabor e assim por diante — não são mantidas sob subdivisão. Uma categoria de objetos que inclui em sua definição apenas propriedades *intrínsecas* é então uma substância ou substantivo de massa; uma classe que inclui *quaisquer* propriedades extrínsecas em sua definição é um substantivo contável. A categoria *Material* é a categoria mais geral de substâncias, que não especifica nenhuma propriedade intrínseca. A categoria *Coisa* é a categoria mais geral de objetos discretos, e não especifica nenhuma propriedade extrínseca.

12.3 EVENTOS

Na Seção 10.4.2, mostramos como o cálculo de situações representa as ações e seus efeitos. O cálculo de situações é limitado em sua aplicabilidade: foi projetado para descrever um mundo em que as ações são discretas, instantâneas e acontecem uma de cada vez. Considere uma ação contínua, como o enchimento de uma banheira. O cálculo de situações pode informar que a banheira está vazia antes da ação e cheia quando a ação é concluída, mas não pode informar o que acontece *durante* a ação. Ele também não pode descrever duas ações acontecendo ao mesmo tempo, tais como escovar os dentes enquanto espera a banheira encher. Para lidar com tais casos, apresentamos um formalismo alternativo conhecido como **cálculo de eventos**, que se baseia em pontos de tempo em vez de situações.[3]

O cálculo de evento reifica fluentes e eventos. O fluente *Em*(*Shankar, Berkeley*) é um objeto que se refere ao fato de Shankar estar em Berkeley, mas por si só não indica se é verdadeiro. Para afirmar que um fluente é realmente verdadeiro em algum ponto no tempo usamos o predicado T, como em $T(Em(Shankar, Berkeley), t)$.

Os eventos são descritos como instâncias de categorias de eventos.[4] O evento E_1 de Shankar voar de São Francisco para Washington, D.C., é descrito como

$$E_1 \in Voos \land Piloto\ (E_1, Shankar) \land Origem\ (E_1, SF) \land Destino\ (E_1, DC).$$

Se estiver muito detalhado, podemos definir uma versão alternativa de três argumentos da categoria dos eventos de voos e dizer

$$E_1 \in Voos\ (Shankar, SF, DC).$$

Em seguida, usamos $Acontece(E_1, i)$ para dizer que o evento E_1 aconteceu no intervalo de tempo i, e dizemos a mesma coisa de forma funcional com $Extensão(E_1) = i$. Representamos os intervalos de tempo pelo par de instantes (início, fim), isto é, $i = (t_1, t_2)$ é o intervalo que começa em t_1 e termina em t_2. O conjunto completo de predicados para uma versão do cálculo de evento é

$T(f, t)$	Fluente f é verdadeiro no instante t
$Acontece(e, i)$	Evento e acontece no intervalo de tempo i
$Inicia(e, f, t)$	Evento e faz com que o fluente f passe a valer no instante t
$Termina(e, f, t)$	Evento e faz com que o fluente f deixe de valer no instante t
$Cortado(f, i)$	Fluente f deixa de ser verdadeiro em algum ponto durante o intervalo de tempo i
$Restaurado\ (f, i)$	Fluente f torna-se verdadeiro em algum momento durante o intervalo de tempo i

Assumimos um evento distinto, *Início*, que descreve o estado inicial informando quais fluentes são iniciados ou terminados no instante inicial. Definimos T dizendo que um fluente é válido em um ponto no tempo se o fluente foi iniciado por um evento em algum momento no passado e não foi tornado falso (cortado) por um evento interveniente. Um fluente não é válido se for terminado por um evento e não se tornou verdadeiro (restaurado) por outro evento. Formalmente, os axiomas são:

$$Acontece(e, (t_1, t_2)) \land Inicia(e, f, t_1) \land \neg Cortado(f\,(\mathrm{t}_1, \mathrm{t})) \land t_1 < t \Rightarrow T(f, t)$$
$$Acontece(e, (t_1, t_2)) \land Termina(e, f, t_1) \land \neg Restaurado(f\,(t_1, t)) \land t_1 < \mathrm{t} \Rightarrow \neg T(f, t)$$

onde *Cortado* e Restaurado são definidos por

$$Cortado(f, (t_1, t_2)) \Leftrightarrow$$
$$\exists\, e, t, t_3\ \mathrm{Acontece}\ (\mathrm{e}, (t, t_3)) \land t_1 \leq t < t_2 \land Termina\ (e, f, t)$$
$$\mathrm{Restaurado}\ (f\,(t_1, t_2)) \Leftrightarrow$$
$$\exists\, e, t, t_3\ \mathrm{Acontece}\ (\mathrm{e}, (t, t_3)) \land t_1 \leq t < t_2 \land Inicia\ (e, f, t).$$

É conveniente estender T para trabalhar em intervalos, bem como em pontos no tempo; um fluente é válido em um intervalo se for válido em todos os pontos dentro do intervalo:

$$T\,(f\,(t_1, t_2)) \Leftrightarrow [\forall\, t\ (t_1 \leq t < t_2) \Rightarrow T\,(f, t)].$$

Os fluentes e as ações são definidos com axiomas de domínio específico que são semelhantes aos axiomas de estados sucessores. Por exemplo, podemos dizer que a única maneira de um agente do mundo de wumpus receber uma seta é no início, e a única maneira de usar uma seta é atirá-la:

$$Inicia\ (e, TemFlecha\ (a), t) \Leftrightarrow e = Início$$
$$Termina\ (e, TemFlecha\ (a), t) \Leftrightarrow e \in Ato\ de\ atirar\ (a).$$

Reificando eventos tornamos possível adicionar qualquer montante de informações arbitrárias sobre eles. Por exemplo, podemos dizer que o voo de Shankar foi acidentado com $Acidentado(E_1)$. Em uma ontologia onde os eventos são predicados n-ários não haveria maneira de adicionar informação extra como essa; mover para um predicado $n + 1$-ário não é uma solução escalável.

Podemos estender o cálculo de eventos para tornar possível representar eventos simultâneos (tal como duas pessoas que são necessárias para montar uma gangorra), os eventos exógenos (tais como o vento soprando e alterando a localização de um objeto), eventos contínuos (tais como o nível de água

na banheira continuamente crescente) e outras complicações.

12.3.1 Processos

Os eventos que vimos até agora são o que chamamos **eventos discretos** — eles têm uma estrutura definida. A viagem de Shankar tem início, meio e fim. Se fosse interrompido no meio caminho, o evento seria algo diferente — não seria uma viagem de San Francisco para Washington, mas uma viagem de San Francisco até algum lugar no Kansas. Por outro lado, a categoria de eventos denotados por *Voos* tem uma qualidade diferente. Se tomarmos um intervalo pequeno do voo de Shankar, digamos, o terceiro período de 20 minutos (enquanto ele espera ansiosamente por um segundo pacote de amendoins), esse evento ainda será um elemento de *Voo*. De fato, isso é verdadeiro para qualquer subintervalo.

As categorias de eventos com essa propriedade são chamadas categorias de **processo** ou categorias de **eventos líquidos**. Qualquer processo *e* que aconteça em um intervalo também acontece sobre qualquer subintervalo.

$$(e \in Processos) \land Acontece\ (e, (t_1, t_4)) \land (t_1 < t_2 < t_3 < t_4) \Rightarrow Acontece\ (e, (t_2, t_3)).$$

A distinção entre eventos líquidos e não líquidos é exatamente análoga à diferença entre substâncias, ou *material*, e objetos individuais, *ou coisas*. De fato, algumas pessoas chamavam os tipos de eventos líquidos de **substâncias temporais**, enquanto itens como manteiga são **substâncias espaciais**.

12.3.2 Intervalos de tempo

O cálculo de evento abre a possibilidade de falar sobre o tempo e intervalos de tempo. Vamos considerar dois tipos de intervalos de tempo: instantes e intervalos estendidos. A distinção é que apenas instantes têm duração zero:

$$Partição(\{Instantes, IntervalosEstendidos\}, Intervalos)$$
$$i \in Instantes \Leftrightarrow Duração(i) = Segundos(0).$$

Em seguida, criamos uma escala de tempo e pontos associados nessa escala a instantes, o que nos fornece tempos absolutos. A escala de tempo é arbitrária; vamos medi-la em segundos e dizer que o instante referente à meia-noite (GMT) em 1º de janeiro de 1900 tem tempo 0. As funções *Início* e *Fim* escolhem os instantes mais antigos e mais recentes em um intervalo, e a função *Instante* entrega o ponto na escala de tempo correspondente a um instante. A função *Duração* fornece a diferença entre a hora final e a hora inicial.

$$Intervalo(i) \Rightarrow Duração(i) = (Hora(Fim(i)) - Hora(Início(i))).$$
$$Instante(Início(DC1900)) = Segundos(0).$$
$$Instante(Início\ (DC2001)) = Segundos(3187324800).$$

Instante(*Início* (*DC*2001)) = *Segundos*(3187324800).
Instante(*Fim*(*DC*2001)) = *Segundos*(3218860800).
Duração(*DC*2001) = *Segundos*(31536000).

Para tornar mais fácil a leitura desses números, também introduzimos uma função *Data*, que recebe seis argumentos (horas, minutos, segundos, dia, mês e ano) e retorna um ponto no tempo (um instante):

Tempo(*Início*(*DC*2001)) = *Data*(0, 0, 0, 1, *janeiro*, 2001)
Data(0, 20, 21, 24, 1, 1995) = *Segundos*(3000000000).

Dois intervalos *Encontram* se o instante final do primeiro é igual ao instante inicial do segundo. O conjunto de relações de intervalo completo, como proposto por Allen (1983), é mostrado graficamente na Figura 12.2 e a seguir logicamente:

$Encontram(i, j) \Leftrightarrow Fim(i) = Início(j)$
$Antes(i, j) \Leftrightarrow Fim(i) < Início(j)$
$Depois(j, i) \Leftrightarrow Antes(i, j)$
$Durante(i, j) \Leftrightarrow Início(j) < Início(i) < Fim(i) < Fim(j)$
$Sobrepõe(i, j) \Leftrightarrow Início(i) < Início(j) < Fim(i) < Fim(j)$
$Inicia(i, j) \Leftrightarrow Início(i) = Início(j)$
$Termina(i, j) \Leftrightarrow Fim(i) = Fim(j)$
$Iguala(i, j) \Leftrightarrow Início(i) = Início(j) \land Fim(i) = Fim(j)$

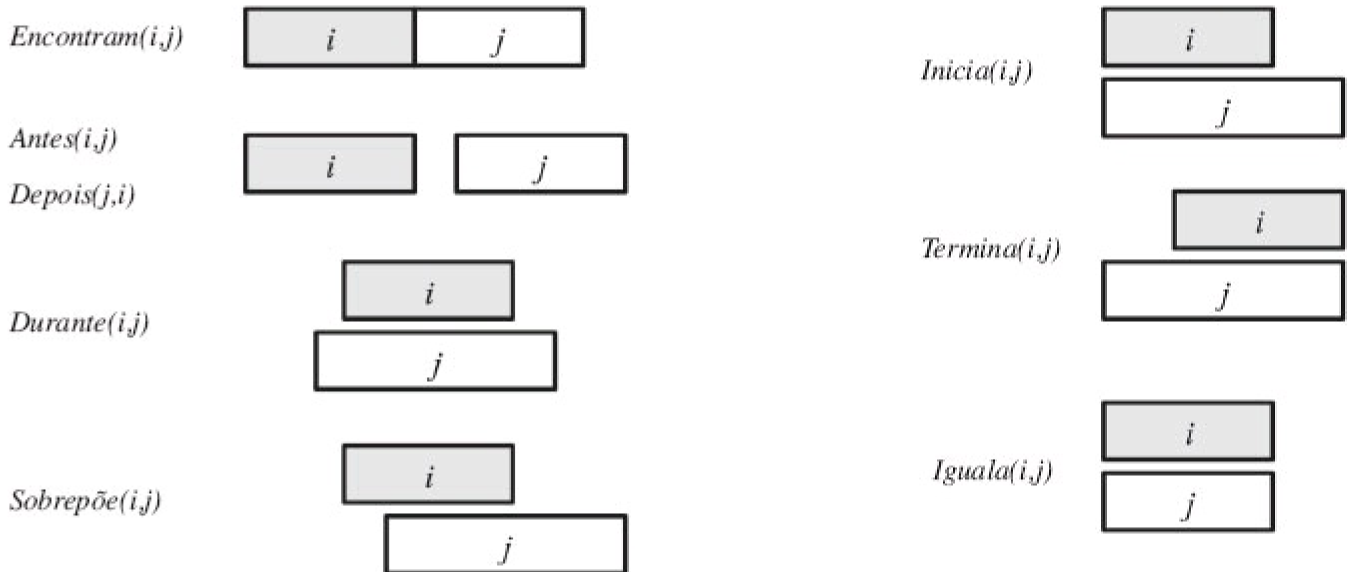


Figura 12.2 Predicados em intervalos de tempo.

Todos eles têm seu significado intuitivo, com exceção de *Sobrepõe*: tendemos a pensar em sobrepor como simétrico (se i se sobrepõe a j, então j se sobrepõe a i), mas nessa definição, *Sobrepõe*(i, j) só será válido se i iniciar antes de j. Para dizer que o reinado de Elizabeth II seguiu o de George VI, e o reinado de Elvis se sobrepôs aos anos 1950, podemos escrever:

Encontram(*ReinadoDe*(*George VI*), *ReinadoDe*(*ElizabethII*)).
Sobrepõe(*Cinquenta*, *ReinadoDe*(*Elvis*)).
Início(*Cinquenta*) = *Início*(*DC*1950).
Fim(*Cinquenta*) = *Fim*(*DC*1959).

12.3.3 Fluentes e objetos

Objetos físicos podem ser visualizados como eventos generalizados, no sentido de que um objeto físico é um bloco de espaço-tempo. Por exemplo, *Estados Unidos* pode ser considerado um evento que começou em, digamos, 1776 como uma união de 13 estados e ainda está em desenvolvimento hoje como uma união de 50 estados. Podemos descrever as mudanças de propriedades dos *Estados Unidos* usando fluentes de estados, tal como *População*(*Estados Unidos*). Outra propriedade dos Estados Unidos que muda a cada quatro ou oito anos, exceto quando ocorre algum infortúnio, é seu presidente. Alguém poderia propor que *Presidente*(*Estados Unidos*) fosse um termo lógico que denotasse um objeto diferente em tempos diferentes. Infelizmente, isso não é possível porque um termo denota exatamente um objeto em dada estrutura de modelo. (O termo *Presidente*(*Estados Unidos*, *t*) pode denotar objetos diferentes, dependendo do valor de *t*, mas nossa ontologia mantém índices de tempo separados de fluentes.) A única possibilidade é que *Presidente*(*Estados Unidos*) denote um único objeto que consiste em diferentes pessoas em diferentes tempos. Esse objeto é George Washington de 1789 até 1797, John Adams de 1797 até 1801, e assim por diante, como na Figura 12.3. Para dizer que George Washington foi presidente ao longo de 1790, podemos escrever

T(*Iguala*(*Presidente*(*Estados Unidos*), *George Washington*), *DC*1790).

Usamos o símbolo da função *Igual a* em vez do predicado lógico padrão = porque não podemos ter um predicado como um argumento para *T* e porque a interpretação *não* é a de que *George Washington* e *Presidente*(*Estados Unidos*) são logicamente idênticos em 1790; a identidade lógica não é algo que possa mudar com o passar do tempo. A identidade está entre os subeventos de cada objeto que são definidos pelo período 1790.

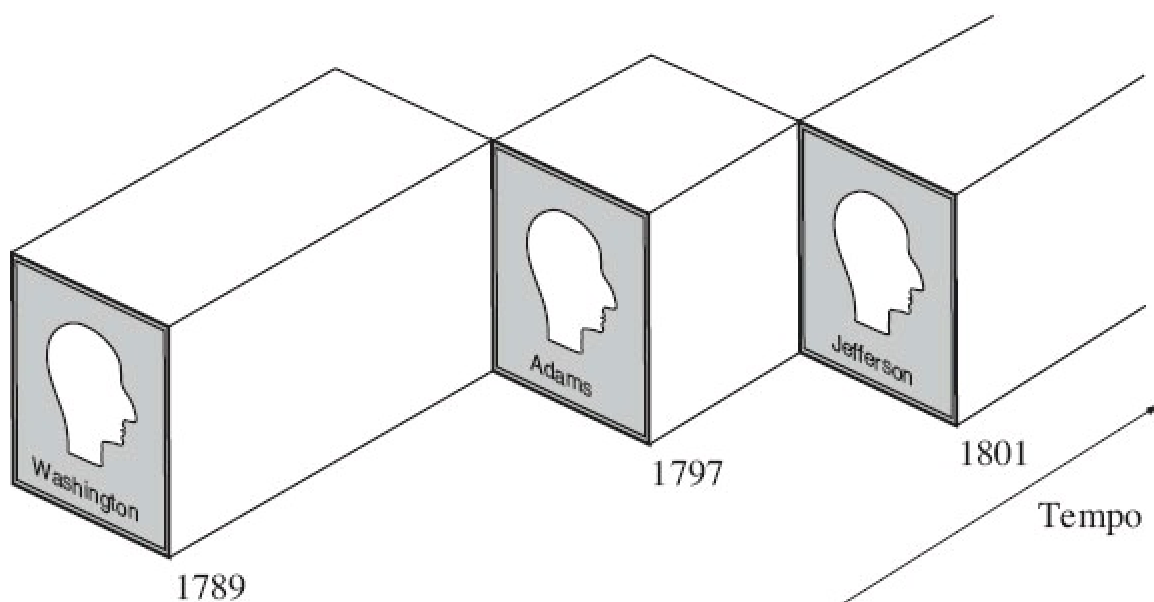


Figura 12.3 Uma visão esquemática do objeto *Presidente*(*Estados Unidos*) para os primeiros 15 anos de sua existência.

12.4 EVENTOS MENTAIS E OBJETOS MENTAIS

Os agentes que construímos até agora têm crenças e podem deduzir novas crenças. Ainda assim nenhum deles tem qualquer conhecimento *sobre* crenças ou *sobre* dedução. O conhecimento sobre o próprio conhecimento e sobre os processos de raciocínio é útil para controlar a inferência. Por exemplo, suponha que Alice pergunte "Qual é a raiz quadrada de 1764" e Bob responda: "Eu não sei." Se Alice insiste, "Pense um pouco mais", Bob deve perceber que, com um pouco mais de raciocínio, essa questão pode de fato ser respondida. Por outro lado, se a questão fosse "Sua mãe está sentada agora?", Bob perceberia que raciocinar mais profundamente não deve ajudar. Conhecimento sobre o conhecimento de outros agentes também é importante; Bob deverá perceber que sua mãe sabe se está sentada ou não, e uma forma de descobrir isso seria lhe perguntando.

O que precisamos ter é um modelo dos objetos mentais que estão na cabeça de alguém (ou na base de conhecimento de alguma coisa) e dos processos mentais que manipulam esses objetos mentais. O modelo não precisa ser detalhado. Não precisamos ter a capacidade de prever quantos milissegundos determinado agente vai demorar para fazer uma dedução. Ficaremos felizes apenas em ser capazes de concluir que a mãe sabe se está sentada ou não.

Começamos com as **atitudes proposicionais** que um agente pode ter em direção aos objetos mentais: atitudes tais como *Acredita*, *Sabe*, *Quer*, *Pretende* e *Informa.* A dificuldade é que essas atitudes não se comportam como predicados "normais". Por exemplo, suponha que nós tentamos afirmar que Lois sabe que o Super-Homem pode voar:

$$Sabe(Lois, PodeVoar(SuperHomem)).$$

Uma questão menor sobre isso é que normalmente pensamos *PodeVoar (SuperHomem)* como uma sentença, mas aqui aparece como um termo. Essa questão pode ser remendada apenas reificando *PodeVoar*(*SuperHomem*), tornando-o um fluente. Um problema mais grave é que, se for verdade que o Super-Homem é o Clark Kent, devemos concluir que Lois sabe que Clark pode voar:

$$(SuperHomem = Clark) \land Sabe(Lois, PodeVoar (SuperHomem))$$
$$\models Sabe (Lois, PodeVoar (Clark)).$$

Essa é uma consequência do fato de que o raciocínio com igualdade é parte da lógica. Normalmente isso é uma coisa boa, se o nosso agente souber que $2 + 2 = 4$ e $4 < 5$, então queremos que o nosso agente saiba que $2 + 2 < 5$. Essa propriedade é chamada de **transparência referencial** — não importa que termo uma lógica use para se referir a um objeto, o que importa é o objeto ao qual o termo dá nome. Mas, para atitudes proposicionais, como *acredita e sabe*, gostaríamos de ter opacidade referencial — os termos usados importam porque nem todos os agentes sabem que termos são correferenciais.

A **lógica modal** foi projetada para resolver esse problema. A lógica regular está preocupada com

uma única modalidade, a da verdade, que nos permite expressar "*P* é verdadeiro". A lógica modal inclui operadores modais especiais que levam sentenças (em vez de termos) como argumentos. Por exemplo, "*A* sabe *P*" é representado com a notação $\mathbf{K}_A P$, onde **K** é o operador modal para conhecimento. Ele recebe dois argumentos, um agente (escrito como subscrito) e uma sentença. A sintaxe da lógica modal é a mesma que a da lógica de primeira ordem, exceto que as sentenças também podem ser formadas com operadores modais.

A semântica da lógica modal é mais complicada. Na lógica de primeira ordem, **um modelo** contém um conjunto de objetos e uma interpretação que mapeia cada nome para o objeto apropriado, relação ou função. Na lógica modal queremos ser capazes de considerar tanto a possibilidade que a identidade secreta do Super-Homem seja Clark como que não seja. Portanto, vamos precisar de um modelo mais complicado, que consista em uma coleção de **mundos possíveis** em vez de apenas um mundo verdadeiro. Os mundos estão ligados em um grafo por **relações de acessibilidade**, uma relação para cada operador modal. Dizemos que o mundo w_1 é acessível do mundo w_0 em relação ao operador modal $\mathbf{K}_A$ se tudo em w_1 for consistente com o que *A* sabe em w_0, e escrevemos como $Acc(\mathbf{K}_A, w_0, w_1)$. Em diagramas como o da Figura 12.4, mostramos a acessibilidade como uma seta entre os mundos possíveis. Como exemplo, no mundo real, Bucareste é a capital da Romênia, mas, para um agente que não sabia disso, outros mundos possíveis são acessíveis, inclusive onde a capital da Romênia seja Sibiu ou Sofia. Presumivelmente, um mundo onde 2 + 2 = 5 não seria acessível a qualquer agente.

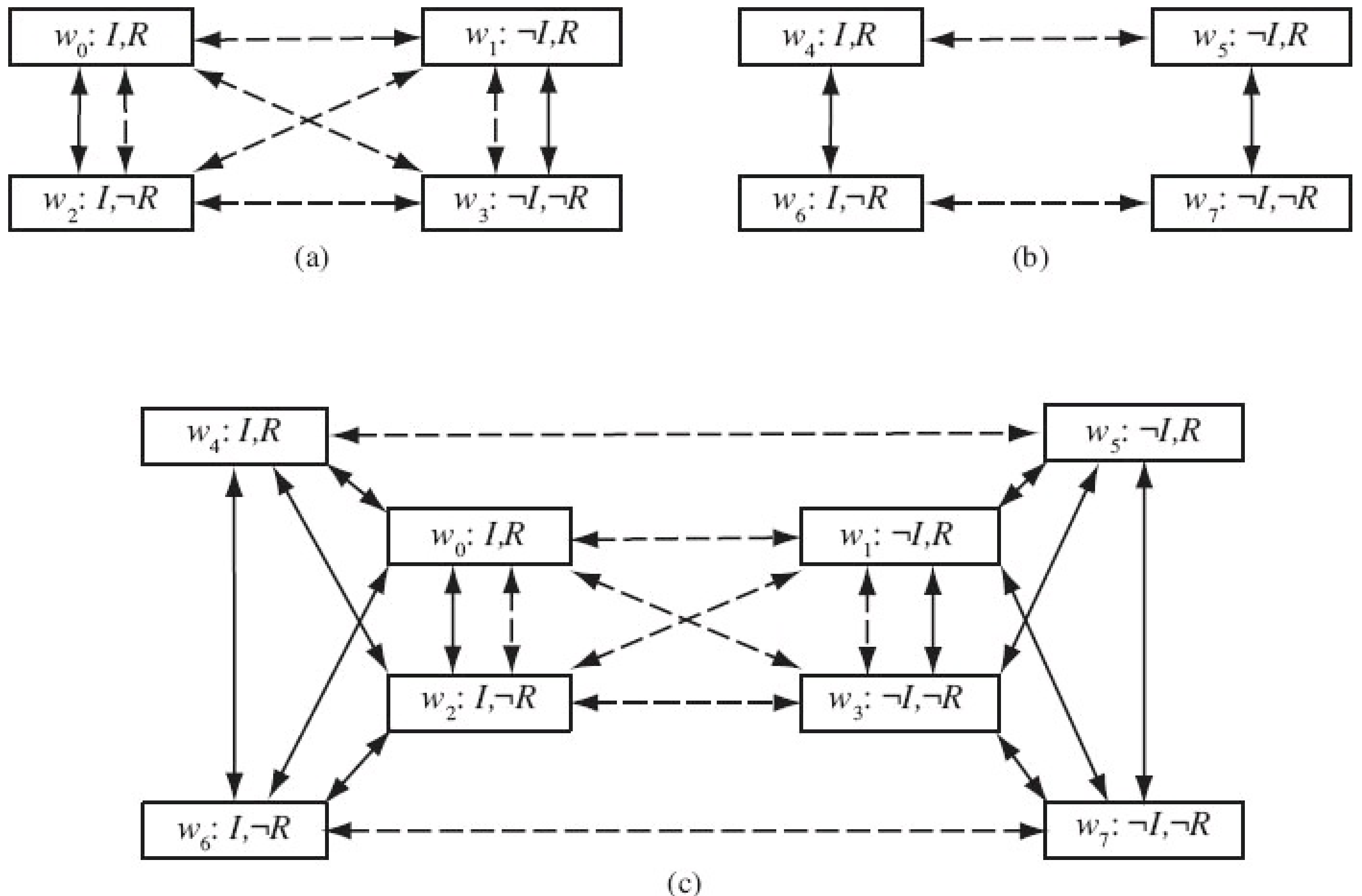


Figura 12.4 Mundos possíveis com relações de acessibilidade $\mathbf{K}_{Super\text{-}homem}$ (setas sólidas) e $\mathbf{K}_{Lois}$ (setas pontilhadas). A proposição *R* significa que "o boletim do tempo para amanhã é de chuva" e *I* significa "a identidade secreta do Super-Homem é Clark Kent". Todos os mundos são acessíveis a

eles mesmos; as setas de um mundo para si mesmo não são mostradas.

Em geral, um átomo de conhecimento $\mathbf{K}_A P$ é verdadeiro no mundo *w* se e somente se *P* for verdadeiro em todo mundo acessível de *w*. A verdade de sentenças mais complexas é derivada pela aplicação recursiva dessa regra e pelas regras normais da lógica de primeira ordem. Isso significa que a lógica modal pode ser usada para raciocinar sobre sentenças de conhecimento aninhadas: o que um agente sabe sobre o conhecimento do outro. Por exemplo, podemos dizer que, apesar de Lois não saber se a identidade secreta do Super-Homem é Clark Kent, ela sabe que Clark sabe:

$$\mathbf{K}_{Lois}\ [\mathbf{K}_{Clark}\ Identidade\ (SuperHomem,\ Clark) \lor \mathbf{K}_{Clark}\ \neg\ Identidade\ (SuperHomem,\ Clark)].$$

A Figura 12.4 mostra alguns mundos possíveis para esse domínio, com relações de acessibilidade para Lois e Super-Homem.

No diagrama superior esquerdo, é do conhecimento comum que o Super-Homem conhece sua própria identidade, e nem ele ou Lois viram o boletim meteorológico. Assim, em w_0 os mundos w_0 e w_2 são acessíveis ao Super--Homem; talvez a previsão seja de chuva, talvez não. Para Lois, os quatro mundos são acessíveis uns aos outros; ela não sabe nada sobre o boletim ou se Clark é o Super-Homem. Mas ela sabe que o Super-Homem sabe se ele é Clark porque em cada mundo que é acessível a Lois ou o Super-Homem conhece *I* ou conhece ¬*I*. Lois não sabe qual seria o caso, mas de qualquer forma sabe que o Super-Homem sabe.

No diagrama superior direito, é de conhecimento comum que Lois viu o boletim meteorológico. Assim, em w_4 ela sabe que está previsto chuva e em w_6 sabe que não está previsto chuva. O Super-Homem não conhece o boletim, mas ele sabe que Lois conhece, porque, em cada mundo acessível para ele, ou ela conhece *R* ou ¬*R*.

No diagrama inferior representamos o cenário em que é de conhecimento comum que o Super-Homem conhece sua identidade, e Lois poderia ou não ter visto o boletim meteorológico. Representamos isso combinando os dois cenários de cima e acrescentando setas para mostrar que o Super-Homem não sabe que cenário realmente é válido. Lois sabe, por isso não precisamos adicionar nenhuma seta para ela. Em w_0, o Super--Homem ainda sabe *I* mas não *R*, e agora ele não sabe se Lois conhece *R*. Pelo que o Super-Homem sabe, ele poderia estar em w_0 ou w_2, e, neste caso Lois não sabe se *R* é verdadeiro, ou poderia estar em w_4, caso ela conheça *R*, ou w_6, caso conheça ¬*R*.

Há um número infinito de mundos possíveis, então o truque é apresentar apenas os que você precisa para representar o que está tentando modelar. Um novo mundo possível é necessário para falar sobre os diferentes fatos possíveis (por exemplo, a chuva está prevista ou não) ou para falar sobre os diferentes estados de conhecimento (por exemplo, Lois sabe que a chuva está prevista). Isso significa que dois mundos possíveis, tais como w_4 e w_0 na Figura 12.4, podem ter os mesmos fatos-base sobre o mundo, mas diferem em suas relações de acessibilidade e, portanto, em fatos sobre o conhecimento.

A lógica modal resolve alguns problemas complicados com a interação dos quantificadores e

conhecimento. A sentença “Bond sabe que alguém é um espião” é ambígua. A primeira leitura é que existe alguém em especial que Bond sabe que é um espião; podemos escrever isso como

$$\exists x\, \mathbf{K}_{Bond} Espião(x),$$

que na lógica modal significa que existe um x que, em todos os mundos acessíveis, Bond sabe ser um espião. A segunda leitura é que Bond só sabe que há pelo menos um espião:

$$\mathbf{K}_{Bond}\ \exists x\, Espião(x).$$

A interpretação lógica modal é que em cada mundo acessível há um x que é um espião, mas não precisa ser o mesmo x em cada mundo.

Agora que temos um operador modal para o conhecimento, podemos escrever axiomas para ele. Em primeiro lugar, podemos dizer que os agentes são capazes de fazer deduções; se um agente sabe P e sabe que P implica Q, então o agente sabe Q:

$$(\mathbf{K}_A P \wedge \mathbf{K}_A (P \Rightarrow Q)) \Rightarrow \mathbf{K}_A Q.$$

A partir daí (e de algumas outras regras sobre identidades lógicas), podemos estabelecer que $\mathbf{K}_A (P \vee \neg P)$ é uma tautologia; cada agente sabe que cada proposição P é verdadeira ou falsa. Por outro lado, $(\mathbf{K}_A P) \vee (\mathbf{K}_A \neg P)$ não é uma tautologia; em geral, haverá um monte de proposições que um agente não sabe se são verdadeiras e não sabe se são falsas.

Diz-se (voltando a Platão) que o conhecimento é uma crença verdadeira justificada. Ou seja, se for verdadeiro, se você acreditar e se tiver uma boa razão que não possa ser contestada, então você sabe. Isso significa que, se você souber alguma coisa, deve ser verdadeira, e temos o axioma:

$$\mathbf{K}_A P \Rightarrow P.$$

Além disso, agentes lógicos devem ser capazes de olhar para o seu próprio conhecimento. Se souberem algo, então eles sabem que sabem disto:

$$\mathbf{K}_A P \Rightarrow \mathbf{K}_A(\mathbf{K}_A P).$$

Podemos definir axiomas semelhantes para a crença (muitas vezes denotada por **B**) e outras modalidades. No entanto, um problema com a abordagem da lógica modal é que ele assume a onisciência lógica na parte dos agentes. Ou seja, se um agente sabe um conjunto de axiomas, então ele sabe todas as consequências desses axiomas. Essa é uma área instável, mesmo para a noção um tanto abstrata de conhecimento, mas parece ainda pior para a crença porque a crença tem mais conotação de se referir a coisas que são representadas fisicamente no agente, e não apenas potencialmente derivável. Houve tentativas de definir uma forma de racionalidade limitada dos agentes para dizer que os agentes acreditam nas afirmações que podem ser derivadas com a aplicação de não mais do que k etapas de raciocínio ou não mais que s segundos de computação. Essas tentativas têm sido geralmente insatisfatórias.

12.5 SISTEMAS DE RACIOCÍNIO PARA CATEGORIAS

Vimos que as categorias são os principais blocos de construção de qualquer esquema de representação de conhecimento em grande escala. Esta seção descreve sistemas especialmente projetados para organizar e raciocinar com categorias. Existem duas famílias de sistemas intimamente relacionadas: as **redes semânticas** oferecem auxílios gráficos para visualização de uma base de conhecimento e algoritmos eficientes para dedução de propriedades de um objeto, de acordo com sua pertinência a uma categoria; as **lógicas de descrição** fornecem uma linguagem formal para construção e combinação de definições de categorias e algoritmos eficientes para definir relacionamentos de subconjuntos e superconjuntos entre categorias.

12.5.1 Redes semânticas

Em 1909, Charles S. Peirce propôs uma notação gráfica de nós e arcos, denominada **grafos existenciais**, que ele chamou de "lógica do futuro". Desse modo, teve início um longo debate entre defensores da "lógica" e defensores de "redes semânticas". Infelizmente, o debate obscureceu o fato de que as redes semânticas — pelo menos aquelas que têm semânticas bem definidas — *são* uma forma de lógica. A notação que as redes semânticas fornecem para certos tipos de sentenças com frequência é mais conveniente, mas, se abstrairmos as questões de "interface humana", os conceitos subjacentes — objetos, relações, quantificação, e assim por diante — serão os mesmos.

Existem muitas variantes de redes semânticas, mas todas são capazes de representar objetos individuais, categorias de objetos e relações entre objetos. Uma notação gráfica típica exibe nomes de objetos ou categorias em elipses ou retângulos e os conecta por meio de arcos rotulados. Por exemplo, a Figura 12.5 tem um arco *ElementoDe* entre *Maria* e *PessoasFemininas*, que corresponde à asserção lógica *Maria* ∈ *PessoasFemininas*; de modo semelhante, o arco *IrmãDe* entre *Maria* e *João* corresponde à asserção *IrmãDe*(*Maria*, *João*). Podemos conectar categorias usando arcos *SubconjuntoDe*, e assim por diante. É tão divertido desenhar bolhas e setas, que podemos nos empolgar. Por exemplo, sabemos que pessoas têm pessoas femininas como mães e, assim, podemos traçar um arco *TemMãe* de *Pessoas* para *PessoasFemininas*? A resposta é não, porque *TemMãe* é uma relação entre uma pessoa e sua mãe, e categorias não têm mães.[5]

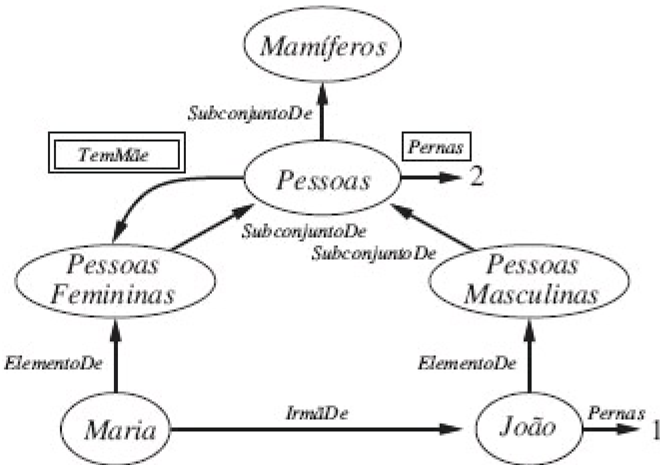


Figura 12.5 Uma rede semântica com quatro objetos (João, Maria, 1 e 2) e quatro categorias. As

relações são denotadas por arcos rotulados.

Por essa razão, usamos uma notação especial — o arco identificado por retângulo de aresta dupla — na Figura 12.5. Esse arco assegura que:

$$\forall x\ x \in Pessoas \Rightarrow [\forall y\ TemMãe(x, y) \Rightarrow y \in PessoasFemininas].$$

Também queremos afirmar que as pessoas têm duas pernas, isto é,

$$\forall x\ x \in Pessoas \Rightarrow Pernas(x, 2).$$

Como antes, precisamos ser cuidadosos para não afirmar que uma categoria tem pernas; o arco de retângulo com arestas simples da Figura 12.5 é usado para afirmar propriedades de todos os elementos de uma categoria.

A notação de rede semântica torna muito conveniente executar o raciocínio de **herança** do tipo introduzido na Seção 12.2. Por exemplo, pelo fato de ser uma pessoa, Maria herda a propriedade de ter duas pernas. Desse modo, para descobrir quantas pernas Maria tem, o algoritmo de herança segue o arco *ElementoDe* desde Maria até a categoria a que ela pertence e depois segue arcos *SubconjuntoDe* pela hierarquia, até encontrar uma categoria para a qual exista um arco *Pernas* identificado por um retângulo — nesse caso, a categoria *Pessoas*. A simplicidade e a eficiência desse mecanismo de inferência, comparado à prova de teoremas lógicos, foi um dos principais fatores de atração das redes semânticas.

A herança fica complicada quando um objeto pode pertencer a mais de uma categoria ou quando uma categoria pode ser um subconjunto de mais de uma outra categoria; isso se chama **herança múltipla**. Em tais casos, o algoritmo de herança pode encontrar dois ou mais valores conflitantes que respondem à consulta. Por essa razão, a herança múltipla foi banida de algumas linguagens de **programação orientada a objetos** (POO), como Java, que utilizam a herança em uma hierarquia de classes. Em geral, isso é permitido em redes semânticas, mas adiaremos a discussão do assunto até a Seção 12.6.

O leitor deve ter notado uma desvantagem óbvia da notação de rede semântica, em comparação com a lógica de primeira ordem: o fato de que arcos entre bolhas representam apenas relações *binárias*. Por exemplo, a sentença *Voar*(*Shankar*, *Nova York*, *NovaDéli*, *Ontem*) não pode ser declarada diretamente em uma rede semântica. Todavia, *podemos* obter o efeito de asserções *n*-árias reificando a proposição em si como um evento pertencente a uma categoria de eventos apropriada. A Figura 12.6 mostra a estrutura de rede semântica para esse evento específico. Note que a restrição para relações binárias força a criação de uma rica ontologia de conceitos reificados.

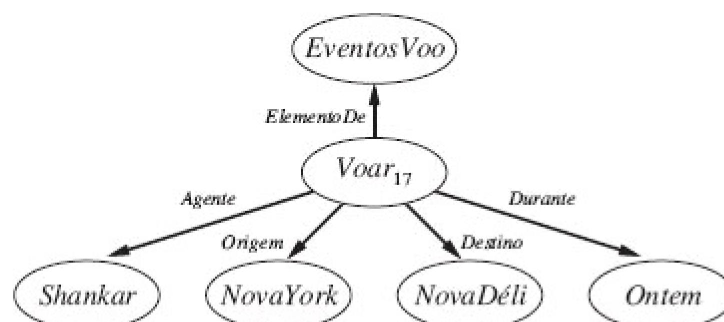


Figura 12.6 Um fragmento de uma rede semântica mostrando a representação da asserção lógica *Voar*(*Shankar*, *NovaYork*, *NovaDéli*, *Ontem*).

A reificação de proposições torna possível representar toda sentença atômica básica livre de funções de lógica de primeira ordem na notação de rede semântica. Certos tipos de sentenças universalmente quantificadas podem ser declaradas com o uso de arcos inversos e com as setas identificadas por retângulos de arestas simples e de arestas duplas aplicadas a categorias, mas isso ainda nos deixa bem longe da lógica de primeira ordem completa. Negação, disjunção, símbolos de funções aninhadas e quantificação existencial — todos estão faltando. Contudo, é *possível* estender a notação para torná-la equivalente à lógica de primeira ordem — como nos grafos existenciais de Peirce —, mas isso nega uma das principais vantagens das redes semânticas: a simplicidade e a transparência dos processos de inferência. Os projetistas podem construir uma grande rede e ainda ter uma boa ideia sobre que consultas serão eficientes porque (a) é fácil visualizar as etapas pelas quais o procedimento de inferência vai passar e (b) em alguns casos, a linguagem de consulta é tão simples que consultas difíceis não podem ser formuladas. Em casos em que a capacidade de expressão se mostra excessivamente limitante, muitos sistemas de redes semânticas fornecem a **conexão procedural** para preencher as lacunas. A conexão procedural é uma técnica por meio da qual uma consulta sobre (ou, às vezes, uma asserção de) certa relação resulta em uma chamada a um procedimento especial projetado para essa relação, e não a um algoritmo de inferência geral.

Um dos aspectos mais importantes das redes semânticas é sua habilidade para representar **valores default** correspondentes a categorias. Examinando a Figura 12.5 cuidadosamente, notamos que João tem uma perna, apesar do fato de ser uma pessoa e de todas as pessoas terem duas pernas. Em uma BC estritamente lógica, isso seria uma contradição; porém, em uma rede semântica, a afirmação de que todas as pessoas têm duas pernas apresenta apenas um *status* default; ou seja, supõe-se que uma pessoa tenha duas pernas, a não ser que isso seja contestado por informações mais específicas. A semântica default é naturalmente imposta pelo algoritmo de herança porque segue arcos ascendentes desde o próprio objeto (João, nesse caso) e para tão logo encontra um valor. Dizemos que o default é **redefinido** pelo valor mais específico. Observe que também poderíamos redefinir o número default de pernas criando uma categoria *PessoasComUmaPerna*, um subconjunto de *Pessoas* ao qual *João* pertence.

Podemos conservar uma semântica estritamente lógica para a rede se dissermos que a asserção *Pernas* para *Pessoas* inclui uma exceção referente a João:

$$\forall x \; x \in Pessoas \land x \neq João \Rightarrow Pernas(x, 2).$$

Para uma rede *fixa*, isso é semanticamente adequado, mas será muito menos conciso que a própria notação de rede, no caso de haver um grande número de exceções. Porém, para uma rede que será atualizada com outras asserções, uma abordagem desse tipo falhará — na realidade, queremos dizer que todas as pessoas com uma única perna, ainda que desconhecidas, também serão exceções. A Seção 12.6 examina com mais profundidade essa questão e o raciocínio default em geral.

12.5.2 Lógicas de descrição

A sintaxe da lógica de primeira ordem foi projetada para facilitar a tarefa de descrever objetos. As **lógicas de descrição** são notações projetadas para tornar mais fácil descrever definições e propriedades de categorias. Os sistemas de lógica de descrição evoluíram a partir das redes semânticas em resposta à pressão para formalizar o que as redes significam enquanto é mantida a ênfase na estrutura taxonômica como um princípio de organização.

As principais tarefas de inferência para lógicas de descrição são a **subordinação (**verificar se uma categoria é um subconjunto de outra pela comparação de suas definições) e a **classificação (**verificar se um objeto pertence a uma categoria). Alguns sistemas também incluem a **consistência** de uma definição de categoria — se os critérios de pertinência são logicamente satisfatíveis.

A linguagem CLASSIC (Borgida *et al*., 1989) é uma lógica de descrição típica. A sintaxe de descrições de CLASSIC é mostrada na Figura 12.7.[6] Por exemplo, para dizer que solteiros são homens adultos não casados, escreveríamos:

Solteiro = *And*(*NãoCasado*, *Adulto*, *Homem*).

O equivalente em lógica de primeira ordem seria:

Solteiro(*x*) ⇔ *NãoCasado*(*x*) ∧ *Adulto*(*x*) ∧ *Homem*(*x*).

Conceito → **Thing** | *NomeConceito*
| **And**(*Conceito*,...)
| **All**(*NomePapel*, *Conceito*)
| **AtLeast**(*Inteiro*, *NomePapel*)
| **AtMost**(*Inteiro*, *NomePapel*)
| **Fills**(*NomePapel*, *NomeIndivíduo*,...)
| **SameAs**(*Caminho*, *Caminho*)
| **OneOf**(*NomeIndivíduo*,...)
Caminho → [*NomePapel*,...]

Figura 12.7 A sintaxe de descrições em um subconjunto da linguagem CLASSIC.

Note que a lógica de descrição tem uma álgebra de operações sobre predicados que certamente não é possível em lógica de primeira ordem. Qualquer descrição em CLASSIC pode ser traduzida em uma sentença equivalente de primeira ordem, mas algumas descrições são mais simples em CLASSIC. Por exemplo, para descrever o conjunto de homens com pelo menos três filhos que estão desempregados e que são casados com médicas e que têm no máximo duas filhas que são todas professoras em departamento de física ou matemática, descreveríamos:

And(*Homem*, *AtLeast*(3, *Filho*), *AtMost*(2, *Filha*),
All(*Filho*, *And*(*Desempregado*, *Casado*, *All*(*Esposa*, *Médica*))),
All(*Filha*, *And*(*Professora*, *Fills*(*Departamento*, *Física*, *Matemática*))))

Deixamos como exercício a conversão dessa expressão para lógica de primeira ordem.

Talvez o aspecto mais importante das lógicas de descrição seja sua ênfase na tratabilidade da inferência. Uma instância de problema é resolvida descrevendo-se a instância e depois indagando-se se ela está subordinada a uma das diversas categorias de soluções possíveis. Em sistemas comuns de lógica de primeira ordem, com frequência, é impossível prever o tempo da solução. Muitas vezes, cabe ao usuário criar a representação necessária para contornar conjuntos de sentenças que parecem estar fazendo o sistema demorar várias semanas para resolver um problema. Por outro lado, a ênfase em lógicas de descrição é assegurar que os testes de subordinação possam ser resolvidos em tempo polinomial em relação ao tamanho das descrições.[7]

Isso parece maravilhoso em princípio, até se perceber que só se pode ter uma dentre duas consequências: ou problemas difíceis não podem ser enunciados de modo algum ou eles exigem descrições exponencialmente extensas! Porém, os resultados de tratabilidade esclarecem que tipos de construções causam problemas e, desse modo, ajudam o usuário a compreender o comportamento de diferentes representações. Por exemplo, normalmente, as lógicas de descrição não têm a *negação* e a *disjunção*. Cada uma força os sistemas de lógica de primeira ordem a passar por uma análise de casos potencialmente exponencial, a fim de assegurar completude. CLASSIC permite apenas uma forma limitada de disjunção nas construções *Fills* e *OneOf*, que tornam possível a disjunção sobre indivíduos explicitamente enumerados, mas não sobre descrições. Com descrições disjuntivas, definições aninhadas podem levar facilmente a um número exponencial de rotas alternativas pelas quais uma categoria pode subordinar outra.

12.6 RACIOCÍNIO COM INFORMAÇÕES DEFAULT

Na seção precedente, vimos um exemplo simples de asserção com *status* de default: as pessoas têm duas pernas. Esse default pode ser anulado por informações mais específicas, como Long John Silver (o pirata) tem uma perna. Vimos que o mecanismo de herança em redes semânticas implementa a redefinição de valores default de uma forma simples e natural. Nesta seção, estudaremos valores default de maneira mais geral, com uma visão voltada para a compreensão da semântica de valores default, e não apenas visando fornecer um mecanismo procedural.

12.6.1 Circunscrição e lógica default

Vimos dois exemplos de processos de raciocínio que violam a propriedade de **monotonicidade** da lógica que foi provada no Capítulo 7.[8] Nesse capítulo vimos que uma propriedade herdada por todos os membros de uma categoria em uma rede semântica podia ser anulada por informação mais específica relativa a uma categoria. Na Seção 9.4.5, vimos que, sob a hipótese do mundo fechado, se uma proposição não for mencionada em BC então $BC \models \neg\alpha$ mas $BC \wedge \alpha \models \alpha$.

A introspecção simples sugere que essas falhas de monotonicidade são muito comuns no raciocínio comum. Parece que os seres humanos frequentemente "saltam para conclusões". Por exemplo, quando alguém vê um carro estacionado na rua, normalmente fica predisposto a acreditar que o automóvel tem quatro rodas, embora só três estejam visíveis. Agora, a teoria da probabilidade

pode sem dúvida fornecer uma conclusão de que a quarta roda existe com probabilidade elevada, ainda que, para a maioria das pessoas, a possibilidade de o carro não ter quatro rodas *não se manifeste, a menos que apareça alguma nova evidência*. Desse modo, parece que a conclusão de haver quatro rodas é alcançada *por default*, na ausência de qualquer razão para colocá-la em dúvida. Se surgirem novas evidências — por exemplo, se alguém vir o proprietário transportando uma roda e notar que o carro está levantado —, então a conclusão poderá ser contestada. Dizemos que essa espécie raciocínio exibe **não monotonicidade** porque o conjunto de crenças não cresce monotonicamente com o tempo à medida que chegam novas evidências. As **lógicas não monotônicas** foram criadas com noções modificadas de verdade e consequência lógica, a fim de captar tal comportamento. Examinaremos duas dessas lógicas que foram extensivamente estudadas: a circunscrição e a lógica default.

A **circunscrição** pode ser vista como uma versão mais poderosa e precisa da hipótese de mundo fechado. A ideia é especificar determinados predicados que consideramos "tão falsos quanto possíveis", isto é, falsos para todo objeto, exceto aqueles para os quais se sabe que eles são verdadeiros. Por exemplo, suponha que desejamos afirmar a regra default de que os pássaros voam. Introduziríamos um predicado, digamos $Anormal_1(x)$, e escreveríamos:

$$Pássaro(x) \land \neg Anormal_1(x) \Rightarrow Voa(x).$$

Se dissermos que $Anormal_1$ deve ser **circunscrito**, um mecanismo de inferência circunscritivo será levado a supor $\neg Anormal_1(x)$, a menos que $Anormal_1(x)$ seja conhecido como verdadeiro. Isso permite que a conclusão $Voa(Tweety)$ seja tirada da premissa $Pássaro(Tweety)$, mas a conclusão não será mais válida se $Anormal_1(Tweety)$ for afirmada.

A circunscrição pode ser visualizada como um exemplo de lógica de **modelo preferencial**. Em tais lógicas, uma sentença é consequência lógica (com *status* default) se é verdadeira em todos os modelos *preferenciais* da BC, em oposição ao requisito de verdade em *todos* os modelos da lógica clássica. No caso da circunscrição, um modelo é preferencial a outro se tem menor número de objetos anormais.[9] Vejamos como essa ideia funciona no contexto de herança múltipla em redes semânticas. O exemplo-padrão em que a herança múltipla é problemática denomina-se "diamante de Nixon" (ou "*Nixon diamond*"). Ele surge a partir da observação de que Richard Nixon era ao mesmo tempo um quacre do inglês "quaker" (e, consequentemente, um pacifista por default) e um republicano (e, consequentemente, um não pacifista por default). Isso pode ser representado da seguinte maneira:

$$Republicano(Nixon) \land Quacre(Nixon).$$
$$Republicano(x) \land \neg Anormal_2(x)) \Rightarrow \neg Pacifista(x).$$
$$Quacre(x) \land \neg Anormal_3(x) \Rightarrow Pacifista(x).$$

Se circunscrevermos $Anormal_2$ e $Anormal_3$, haverá dois modelos preferenciais: um modelo em que $Anormal_2(Nixon)$ e $Pacifista(Nixon)$ são válidas e outro em que $Anormal_3(Nixon)$ e $\neg Pacifista(Nixon)$ são válidas. Desse modo, o mecanismo de inferência circunscritivo permanece corretamente agnóstico sobre o fato de Nixon ser ou não pacifista. Se desejarmos, além disso,

afirmar que as crenças religiosas devem ter precedência sobre as crenças políticas, poderemos empregar um formalismo chamado **circunscrição priorizada** para dar preferência a modelos em que $Anormal_3$ é minimizada.

A **lógica default** é um formalismo em que podem ser escritas **regras default** para gerar conclusões contingentes e não monotônicas. Uma regra default é semelhante a:

$$Pássaro(x) : Voa(x)/Voa(x).$$

Essa regra significa que, se $Pássaro(x)$ é verdadeira e se $Voa(x)$ é consistente com a base de conhecimento, então $Voa(x)$ pode ser concluída por default. Em geral, uma regra default tem a forma:

$$P : J_1, \ldots, J_n/C$$

onde P é chamado pré-requisito, C é a conclusão e J_i são as justificativas — se for possível provar que qualquer uma delas é falsa, então a conclusão não poderá ser derivada. Qualquer variável que aparecer em J_i ou C também terá de aparecer em P. O exemplo do dilema de Nixon pode ser representado em lógica default com um fato e duas regras default:

$$Republicano(Nixon) \wedge Quacre(Nixon).$$
$$Republicano(x) : \neg Pacifista(x) \ / \ \neg Pacifista(x).$$
$$Quacre(x) : Pacifista(x) \ / \ Pacifista(x).$$

Para interpretar o que significam as regras default, definimos a noção de **extensão** de uma teoria default como um conjunto máximo de consequências da teoria. Isto é, uma extensão S consiste nos fatos conhecidos originais e em um conjunto de conclusões das regras default, tais que nenhuma conclusão adicional possa ser obtida de S e que as justificativas de toda conclusão default em S sejam consistentes com S. Como no caso dos modelos preferidos na circunscrição, temos duas extensões possíveis para o dilema de Nixon: uma em que ele é pacifista e uma em que ele não é pacifista. Existem esquemas de priorização nos quais algumas regras default podem ter precedência sobre outras, permitindo que algumas ambiguidades sejam resolvidas.

Desde 1980, quando as lógicas não monotônicas foram propostas pela primeira vez, houve um grande progresso na compreensão de suas propriedades matemáticas. Porém ainda existem questões não resolvidas. Por exemplo, se “Os carros têm quatro rodas” é falso, o que significa ter essa asserção em uma base de conhecimento? Que conjunto de regras default seria interessante ter? Se não pudermos decidir, para cada regra separadamente, se ela pertence à nossa base de conhecimento, então teremos um sério problema de falta de modularidade. Por fim, como as crenças que têm *status* de default podem ser usadas na tomada de decisões? Provavelmente, essa é a questão mais difícil para o raciocínio default. Com frequência, decisões envolvem compromissos, e portanto é necessário comparar as *intensidades* de crença nos resultados de diferentes ações e o custo de tomar uma decisão errada. Nos casos em que as mesmas espécies de decisões estão sendo tomadas repetidamente, é possível interpretar regras default como declarações de “probabilidade de limiar”. Por exemplo, a regra default “Meus freios estão sempre OK” na realidade significa “A probabilidade de que meus freios estejam OK, não sendo dada nenhuma outra informação, é suficientemente alta

para que a decisão ótima no meu caso seja dirigir sem verificá-los". Quando o contexto de decisão se altera — por exemplo, quando se está dirigindo um caminhão pesadamente carregado montanha abaixo em uma estrada íngreme —, a regra default se torna repentinamente inadequada, embora não exista nenhuma evidência a sugerir que os freios estão defeituosos. Essas considerações levaram alguns pesquisadores a refletir sobre como incorporar o raciocínio com defaults dentro da teoria da probabilidade ou da utilidade.

12.6.2 Sistemas de manutenção de verdade

Vimos que muitas das inferências derivadas por um sistema de representação de conhecimento só terão *status* de default, em vez de estarem absolutamente certas. Inevitavelmente, alguns desses fatos deduzidos se mostrarão incorretos e terão de ser reconsiderados em face de novas informações. Esse processo é chamado **revisão de crenças**.[10] Suponha que uma base de conhecimento *BC* contenha uma sentença *P* — talvez uma conclusão default registrada por um algoritmo de encadeamento para a frente ou talvez apenas uma asserção incorreta — e queremos executar *TELL*(*BC*, ¬*P*). Para evitar criar uma contradição, primeiro devemos executar *RETRACT*(*BC*, *P*). Parece bem fácil.

No entanto, surgiriam problemas se quaisquer sentenças *adicionais* fossem deduzidas a partir de *P* e afirmadas na BC. Por exemplo, a implicação $P \Rightarrow Q$ poderia ter sido usada para adicionar *Q*. A "solução" óbvia — reconsiderar todas as sentenças deduzidas a partir de *P* — falhará porque tais sentenças podem ter outras justificativas além de *P*. Por exemplo, se *R* e $R \Rightarrow Q$ também estiverem na BC, então *Q* não terá de ser removida. Os **sistemas de manutenção de verdade**, ou TMSs (do inglês "Truth Maintenance Systems"), foram projetados para manipular exatamente esses tipos de complicações.

Uma abordagem muito simples para manutenção de verdade é manter o controle da ordem em que as sentenças são apresentadas à base de conhecimento, numerando-as de P_1 até P_n. Quando a chamada RETRACT(*BC*, P_i) é feita, o sistema reverte ao estado imediatamente anterior à adição de P_i, removendo tanto P_i quanto quaisquer inferências que tenham sido derivadas de P_i. As sentenças P_{i+1} até P_n podem então ser novamente adicionadas. Isso é simples e garante que a base de conhecimento será consistente, mas a reconsideração de P_i exige a retirada e a reafirmação de $n - i$ sentenças, além de ser preciso desfazer e refazer todas as inferências obtidas a partir dessas sentenças. Em sistemas aos quais estão sendo adicionados muitos fatos — como grandes bancos de dados comerciais —, isso é impraticável.

Uma abordagem mais eficiente é o sistema de manutenção de verdade baseado em justificativa, ou **JTMS**. Em um JTMS, cada sentença na base de conhecimento é anotada com uma **justificativa** que consiste no conjunto de sentenças a partir das quais ela foi deduzida. Por exemplo, se a base de conhecimento já contém $P \Rightarrow Q$, então TELL(*P*) fará *Q* ser adicionada com a justificativa $\{P, P \Rightarrow Q\}$. Em geral, uma sentença pode ter qualquer número de justificativas. As justificativas tornam a retração eficiente. Dada a chamada RETRACT(*P*), o JTMS eliminará exatamente as sentenças para as quais *P* é um elemento de toda justificativa. Assim, se uma sentença *Q* tivesse a única justificativa $\{P, P \Rightarrow Q\}$, ela seria removida; se tivesse a justificativa adicional $\{P, P \lor R \Rightarrow Q\}$, ela ainda

seria removida; mas, se também tivesse a justificativa $\{R, P \vee R \Rightarrow Q\}$, ela seria poupada. Desse modo, o tempo necessário para a retração de P depende apenas do número de sentenças derivadas de P, e não do número de outras sentenças adicionadas desde que P entrou na base de conhecimento.

O JTMS pressupõe que as sentenças que são consideradas uma vez provavelmente serão consideradas de novo; assim, em vez de eliminar inteiramente uma sentença da base de conhecimento quando ela perder todas as justificativas, simplesmente marcamos a sentença para indicar que ela está *fora* da base de conhecimento. Se uma asserção subsequente restaurar uma das justificativas, marcaremos a sentença indicando que ela está *dentro* outra vez. Desse modo, o JTMS preserva todas as cadeias de inferência que utiliza e não precisa derivar novamente as sentenças quando uma justificativa se torna válida de novo.

Além de manipular a retirada de informações incorretas, os TMSs podem ser usados para acelerar a análise de várias situações hipotéticas. Por exemplo, suponha que o comitê olímpico da Romênia esteja escolhendo locais para os eventos de natação, atletismo e equitação dos jogos de 2048 a serem realizados na Romênia. Por exemplo, seja a primeira hipótese *Local*(*Natação*, *Pitesti*), *Local*(*Atletismo*, *Bucareste*) e *Local*(*Equitação*, *Arad*). É necessário bastante raciocínio para determinar as consequências logísticas e, portanto, o interesse dessa seleção. Se, em vez disso, quisermos considerar *Local*(*Atletismo*, *Sibiu*), o TMS evitará a necessidade de começar de novo desde o início. Nesse caso, simplesmente iremos retirar *Local*(*Atletismo*, *Bucareste*) e afirmar *Local*(*Atletismo*, *Sibiu*), e o TMS cuidará das revisões necessárias. As cadeias de inferência geradas a partir da escolha de Bucareste poderão ser reutilizadas com Sibiu, desde que as conclusões sejam as mesmas.

Um sistema de manutenção de verdade baseado em hipóteses, ou **ATMS**, foi projetado para tornar particularmente eficiente esse tipo de troca de contexto entre mundos hipotéticos. Em um JTMS, a manutenção de justificativas permite a rápida movimentação de um estado para outro fazendo-se algumas retiradas e asserções, mas, em qualquer instante, apenas um estado é representado. Um ATMS representa ao mesmo tempo *todos* os estados que já foram considerados. Enquanto um JTMS simplesmente identifica cada sentença como *dentro* ou *fora*, um ATMS controla, para cada sentença, que hipóteses tornariam a sentença verdadeira. Em outras palavras, cada sentença tem um rótulo que consiste em um conjunto de conjuntos de hipóteses. A sentença é válida apenas nos casos em que todas as hipóteses de um dos conjuntos de hipóteses são válidas.

Os sistemas de manutenção de verdade também fornecem um mecanismo para gerar **explicações**. Tecnicamente, uma explicação de uma sentença P é um conjunto de sentenças E tal que E tem P como consequência lógica. Se já soubermos que as sentenças contidas em E são verdadeiras, então E simplesmente fornecerá uma base suficiente para provar que P deve ocorrer. Porém, as explicações também podem incluir **hipóteses** — sentenças que não sabemos se são verdadeiras, mas que seriam suficientes para provar P se fossem verdadeiras. Por exemplo, poderíamos não ter informações suficientes para provar que o carro de alguém não dá partida, mas uma explicação razoável poderia incluir a hipótese de a bateria estar descarregada. Essa hipótese, combinada com o conhecimento de como os carros operam, explica o não comportamento observado. Na maioria dos casos, preferiremos uma explicação E que seja minimal, significando que não existe nenhum subconjunto próprio de E que também seja uma explicação. Um ATMS pode gerar explicações para o problema do "carro que não dá partida" fazendo suposições (como "carro tem gasolina" ou "bateria

descarregada”) em qualquer ordem que desejarmos, mesmo que algumas hipóteses sejam contraditórias. Em seguida, examinaremos o rótulo correspondente à sentença “carro não dá partida” para examinar os conjuntos de hipóteses que justificariam a sentença.

Os algoritmos exatos utilizados para implementar os sistemas de manutenção de verdade são um pouco complicados, e não os abordaremos aqui. A complexidade computacional do problema de manutenção de verdade é pelo menos tão grande quanto a da inferência proposicional, isto é, NP-difícil. Portanto, você não deve esperar que a manutenção de verdade seja uma panaceia. Porém, quando utilizado com cuidado, um TMS pode proporcionar aumento substancial na habilidade de um sistema lógico para tratar ambientes e hipóteses complexas.

12.7 O MUNDO DE COMPRAS DA INTERNET

Nesta seção final juntamos tudo o que aprendemos para codificar o conhecimento para um agente de pesquisa de compras que ajuda um comprador a encontrar ofertas de produtos na Internet. O agente de compras recebe uma descrição do produto feita pelo comprador e tem a tarefa de produzir uma lista de páginas da Web que oferecem tal produto à venda classificando quais ofertas são melhores. Em alguns casos, a descrição do produto do comprador será precisa, como em *câmera digital Canon Rebel XTi*, e a tarefa será descobrir a(s) loja(s) com a melhor oferta. Em outros casos, a descrição será especificada apenas parcialmente, como em câmera digital por menos de *$300*, e o agente terá de comparar diferentes produtos.

O ambiente do agente de compras é a World Wide Web em toda a sua complexidade — não um miniambiente simulado. As percepções do agente são páginas Web mas, enquanto um usuário humano da Web veria páginas exibidas como um array de pixels na tela, o agente de compras perceberá uma página como uma cadeia de caracteres que consiste em palavras comuns entremeadas com comandos de formatação na linguagem de marcação HTML. A Figura 12.8 mostra uma página da Web e uma cadeia de caracteres HTML correspondente. O problema de percepção para o agente de compras envolve a extração de informações úteis de percepções desse tipo.

Exemplo de Loja On-line

Selecione em nossa excelente linha de produtos:

- Computadores
- Câmeras
- Livros
- Vídeos
- Música

```
<h1>Exemplo de Loja On-line</h1>
<i>Selecione</i> em nossa excelente linha de produtos:
<ul>
<li> <a href=”http://example.com/compu”>Computadores</a>
```

```
<li> <a href="http://example.com/camer">Câmeras</a>
<li> <a href="http://example.com/livros">Livros</a>
<li> <a href="http://example.com/video">Vídeos</a>
<li> <a href="http://example.com/music">Música</a>
</ul>
```

Figura 12.8 A página da Web de uma loja on-line genérica na forma percebida pelo usuário humano de um navegador (parte superior) e a cadeia de texto em HTML correspondente conforme é percebida pelo navegador ou pelo agente de compras (parte inferior). Em HTML, caracteres entre < e > são diretivas de marcação que especificam como a página é exibida. Por exemplo, a cadeia <i>Selecione</i> significa alternar para fonte em itálico, exibir a palavra *Selecionar* e depois encerrar o uso da fonte em itálico. Um identificador de página como http://example.com/livros é chamado URL (Uniform Resource Locator — localizador de recursos uniforme). A marcação *Livros* significa criar um link de hipertexto para *url* com o **texto de âncora** *Livros*.

Sem dúvida, a percepção nas páginas da Web é mais fácil que, digamos, a percepção enquanto se dirige um táxi no Cairo. Todavia, existem complicações na tarefa de percepção na Internet. A página Web da Figura 12.8 é muito simples em comparação com sites de compras reais, que podem incluir CSS, cookies, Java, JavaScript, Flash, protocolos de exclusão de robôs, HTML mal formada, arquivos de som, filmes e texto que só aparece como parte de uma imagem JPEG. Um agente que pode lidar com toda a Internet é quase tão complexo quanto um robô que pode se mover no mundo real. Vamos nos concentrar em um agente simples que ignora a maior parte dessas complicações.

A primeira tarefa do agente é encontrar ofertas de produtos relevantes para a consulta. Se a consulta for "notebooks", uma página da Web com uma resenha dos notebooks mais recentes de tecnologia de ponta seria relevante, mas, se não fornecer uma maneira de comprar, não é uma oferta. Por ora, podemos dizer que uma página é uma oferta se contiver as palavras "comprar" ou "preço" ou "adicionar ao carrinho" dentro de um link HTML ou formulário na página. Por exemplo, se a página contiver uma cadeia da forma "<a… adicionar ao carrinho… </ a", é uma oferta. Isso poderia ser representado em lógica de primeira ordem, mas é mais simples codificá-lo em código de programa. Mostraremos como fazer a extração de informações mais sofisticadas na Seção 22.4.

12.7.1 Seguindo links

A estratégia é começar na homepage de uma loja on-line e examinar todas as páginas que podem ser alcançadas seguindo-se links relevantes.[11] O agente terá conhecimento de várias lojas, por exemplo:

$$Amazon \in LojasOn\text{-}line \wedge Homepage(Amazon, \text{“amazon.com”})$$
$$Ebay \in LojasOn\text{-}line \wedge Homepage(Ebay, \text{“ebay.com”})$$
$$LojaExemplo \in LojasOn\text{-}line \wedge Homepage\ (LojaExemplo, \text{“exemplo.com”}).$$

Essas lojas classificam suas mercadorias em categorias de produtos e fornecem links para as

categorias importantes a partir de sua homepage. As categorias secundárias podem ser alcançadas seguindo-se uma cadeia de links relevantes e, eventualmente, chegaremos a ofertas. Em outras palavras, uma página é relevante para a consulta se pode ser alcançada por meio de uma cadeia de zero ou mais links de categorias relevantes a partir da homepage de uma loja, e depois seguindo-se mais um link para a oferta do produto. Podemos definir a relevância:

$$\begin{aligned}
&Relevante(página, consulta) \Leftrightarrow \\
&\quad \exists\ loja, home\ loja \in LojasOn\text{-}line \wedge Homepage(loja, home) \\
&\quad \wedge \exists\ url, url_2\ CadeiaRelevante(home, url_2, consulta) \wedge Link(url_2, url) \\
&\quad\quad \wedge\ página = Conteúdo(url).
\end{aligned}$$

Aqui, o predicado *Link*(*de*, *para*) significa que existe um hiperlink do URL de até o URL *para*. Para definir o que é considerado uma *CadeiaRelevante*, precisamos seguir não apenas hiperlinks antigos, mas somente os links cujo texto de âncora associado indicam que o link é relevante para a consulta de produtos. Para isso, usaremos *TextoLink*(*de*, *para*, *texto*) para indicar que existe um link entre *de* e *para* contendo *texto* como texto de âncora. Uma cadeia de links entre dois URLs, início e fim, é relevante para uma descrição *d* se o texto de âncora de cada link é um nome de categoria relevante para *d*. A existência da própria cadeia é determinada por uma definição recursiva, com a cadeia vazia (*início* = *fim*) sendo o caso básico:

$$\begin{aligned}
&CadeiaRelevante(início, fim, consulta) \Leftrightarrow (início = fim) \\
&\quad \vee (\exists\ u, texto\ TextoLink(início, u, texto) \wedge NomeCategoriaRelevante(consulta, texto) \\
&\quad\quad \wedge CadeiaRelevante(u, fim, consulta)).
\end{aligned}$$

Agora, devemos definir o que significa o fato de o *texto* ser um *NomeCategoriaRelevante* para *consulta*. Primeiro, precisamos relacionar as cadeias às categorias que elas identificam. Isso é feito usando-se o predicado *Nome*(*s*, *c*), que nos informa que a cadeia *s* é um nome para a categoria *c* — por exemplo, poderíamos afirmar que *Nome*(“*laptops*”, Laptops). Temos mais alguns exemplos do predicado Nome na Figura 12.9(b). Em seguida, definimos a relevância. Suponha que a *consulta* seja “laptops”. Então, *NomeCategoriaRelevante*(*consulta*, *texto*) é verdadeira quando um dos itens a seguir é válido:

- O *texto* e a *consulta* identificam a mesma categoria — por exemplo, “notebooks” e “laptops”.
- O *texto* identifica uma supercategoria como “computadores”.
- O *texto* identifica uma subcategoria, tal como “notebooks ultraleves”.

Livros ⊂ *Produtos*
GravaçõesMúsica ⊂ *Produtos*
  CDsMúsica ⊂ *GravaçõesMúsica*
Eletrônicos ⊂ *Produtos*
  CâmerasDigitais ⊂ *Eletrônicos*
  EquipamentosEstéreo ⊂ *Eletrônicos*
  Computadores ⊂ *Eletrônicos*
    ComputadoresDesktops ⊂ *Computadores*
    ComputadoresLaptops ⊂ *Computadores*
    ...

(a)

Nome(*"livros", Livros*)
Nome(*"música", GravaçõesMúsica*)
  Nome(*"CDs", CDsMúsica*)
Nome(*"eletrônicos", Eletrônicos*)
  Nome(*"câmerasdigitais", CâmerasDigitais*)
  Nome(*"estéreos", EquipamentosEstéreo*)
  Nome(*"computadores", Computadores*)
    Nome(*"desktops", ComputadoresDesktops*)
    Nome(*"Laptops", ComputadoresLaptops*)
    Nome(*"notebooks", ComputadoresLaptops*)
    ...

(b)

Figura 12.9 (a) Taxonomia de categorias de produtos. (b) Nomes para essas categorias.

A definição lógica de *NomeCategoriaRelevante* é:

$$NomeCategoriaRelevante(consulta, texto) \Leftrightarrow$$
$$\exists\, c_1, c_2\ Nome(consulta, c_1) \wedge Nome(texto\ , c_2) \wedge (c_1 \subseteq c_2 \vee c_2 \subseteq c_1). \qquad (12.1)$$

Caso contrário, o texto de âncora é irrelevante porque identifica uma categoria fora dessa linha, como "roupas" ou "gramado & jardim".

Então, para seguir links relevantes é essencial ter uma hierarquia rica de categorias de produtos. A parte superior dessa hierarquia poderia ser semelhante à da Figura 12.9(a). Não será possível listar todas as categorias de compras possíveis porque um comprador sempre poderia externar algum novo desejo e os fabricantes sempre lançarão novos produtos para satisfazer esses desejos (aquecedores elétricos para os joelhos?). Todavia, uma ontologia de cerca de mil categorias servirá como uma ferramenta muito útil para a maioria dos compradores.

Além da hierarquia de produtos em si, também precisamos ter um rico vocabulário de nomes para categorias. A vida seria muito mais fácil se houvesse uma correspondência de um para um entre as categorias e as cadeias de caracteres que as identificam. Já vimos o problema da **sinonímia** — dois nomes para a mesma categoria, como "computadores laptops" e "laptops". Também existe o problema da **ambiguidade** — um único nome para duas ou mais categorias distintas. Por exemplo, se adicionarmos a sentença

Nome(*"CDs", CertificadosDeDepósito*)

à base de conhecimento da Figura 12.9(b), "CDs" vai identificar duas categorias diferentes.

A sinonímia e a ambiguidade podem provocar um aumento significativo no número de caminhos que o agente tem de seguir e, às vezes, podem tornar difícil definir se determinada página é de fato relevante. Um problema muito mais sério é o fato de existir uma variedade muito ampla de descrições que um usuário pode digitar e de nomes de categorias que uma loja pode usar. Por exemplo, o link poderia informar "laptop" quando a base de conhecimento tem apenas "laptops" ou, então, talvez o usuário quisesse procurar "um computador que eu possa colocar na mesinha retrátil de um assento da classe econômica de um avião". É impossível enumerar com antecedência todos os possíveis modos de identificar uma categoria e, assim, o agente terá de ser capaz de efetuar certo

raciocínio adicional em alguns casos para determinar se a relação *Nome* é válida. No pior caso, isso exigirá compreensão completa de linguagem natural, um tópico que adiaremos até o Capítulo 22. Na prática, algumas regras simples — como permitir que “laptop” corresponda a uma categoria denominada “laptops” — alcançam bons resultados. O Exercício 12.10 lhe pede para desenvolver um conjunto de tais regras depois de fazer alguma pesquisa em lojas on-line.

Dadas as definições lógicas dos parágrafos precedentes e bases de conhecimentos adequadas de categorias de produtos e convenções de nomenclatura, estamos prontos para aplicar um algoritmo de inferência para obter um conjunto de ofertas relevantes para nossa consulta? Não exatamente! O elemento que falta é a função *Conteúdo*(*url*), que se refere à página HTML em dado URL.O agente não tem o conteúdo da página de todo URL em sua base de conhecimento; ele também não tem regras explícitas para deduzir qual poderia ser esse conteúdo. Em vez disso, podemos organizar tudo para que o procedimento de HTTP correto seja executado sempre que um subobjetivo envolver a função *Conteúdo*. Dessa maneira, para o mecanismo de inferência será como se a Web inteira estivesse dentro da base de conhecimento. Esse é um exemplo de técnica geral chamada **conexão procedural**, por meio da qual predicados e funções específicos podem ser tratados por métodos de uso especial.

12.7.2 Comparação entre ofertas

Vamos supor que os processos de raciocínio da seção precedente tenham produzido um conjunto de páginas de ofertas para nossa consulta “laptops”. Para comparar essas ofertas, o agente deve extrair as informações relevantes — preço, velocidade, tamanho de disco, peso, e assim por diante — das páginas de ofertas. Essa pode ser uma tarefa difícil no caso das páginas da Web reais, por todas as razões mencionadas anteriormente. Um modo comum de lidar com esse problema é usar programas chamados **envoltórios** (“wrappers”) para extrair informações de uma página. A tecnologia de extração de informações é discutida na Seção 22.4. No momento, supomos que os envoltórios existem e, quando são dadas uma página e uma base de conhecimento, eles acrescentam asserções à base de conhecimento. Em geral, uma hierarquia de envoltórios seria aplicada a uma página: um envoltório muito geral para extrair datas e preços, outro mais específico para extrair atributos referentes a produtos relacionados a computadores e, se necessário, um envoltório específico do site que conheça o formato de determinada loja. Dada uma página no site exemplo.com com o texto

```
IBM ThinkBook 970. Nosso preço: US$399,00
```

seguido por diversas especificações técnicas, gostaríamos de ter um envoltório para extrair informações como as seguintes:

$$\exists\, c, oferta\; c \in ComputadoresLaptop \land oferta \in ProdutosOferecidos \land$$
$$Fabricante\,(c, IBM) \land Modelo\,(c, ThinkBook970) \land$$
$$TamanhoDaTela(c, Polegadas(14)) \land TipoDaTela(c, ColorLCD) \land$$
$$TamanhoDaMemoria(c, Gigabytes(2)) \land VelocidadeDaCPU(c, GHz(1{,}2)) \land$$
$$ProdutoOferecido(oferta, c) \land Loja(oferta, GenStore) \land$$
$$URL(oferta, \text{“}\texttt{exemplo.com/computadores/34356.html}\text{”}) \land$$
$$Preço(oferta, \$(399)) \land Data(oferta, Hoje).$$

Esse exemplo ilustra várias questões que surgem quando levamos a sério a tarefa de engenharia de conhecimento para transações comerciais. Por exemplo, note que o preço é um atributo da *oferta*, não do produto em si. Isso é importante porque a oferta em determinada loja pode mudar a cada dia, até para o mesmo laptop individual; para algumas categorias — como casas e pinturas —, o mesmo objeto individual pode ser até oferecido ao mesmo tempo por diferentes intermediários com preços distintos. Existem ainda outras complicações de que não tratamos, como a possibilidade de o preço depender do método de pagamento e das qualificações do comprador para obter certos descontos. A tarefa final é comparar as ofertas que foram extraídas. Por exemplo, considere estas três ofertas:

A : 1.4 GHz CPU, 2GB RAM, 250 GB disk, US$299,00.
B : 1.2 GHz CPU, 4GB RAM, 350 GB disk, US$500,00.
C : 1.2 GHz CPU, 2GB RAM, 250 GB disk, US$399,00.

C é **dominado** por *A*; isto é, *A* é mais barato e mais rápido e, fora isso, eles são idênticos. Em geral, *X* domina *Y* se *X* tem um valor melhor em pelo menos um atributo e não é pior em qualquer atributo. Porém, nem *A* nem *B* dominam um ao outro. Para decidir qual deles é melhor, precisamos saber como o comprador avalia a velocidade da CPU e o preço em comparação com memória e espaço em disco. O tópico geral de preferências entre vários atributos é examinado na Seção 16.4; por enquanto, nosso agente de compras simplesmente retornará uma lista de todas as ofertas não dominadas que satisfazem à descrição do comprador. Nesse exemplo, *A* e *B* são não dominados. Note que esse resultado se baseia na suposição de que todo mundo prefere preços mais baixos, processadores mais rápidos e mais espaço de armazenamento. Alguns atributos, como tamanho de tela em um notebook, dependem da preferência específica do usuário (portabilidade *versus* visibilidade); nesses casos, o agente de compras simplesmente terá de perguntar ao usuário.

O agente de compras que descrevemos aqui é simples; são possíveis muitos refinamentos. Ainda assim, ele tem capacidade suficiente para poder, com o conhecimento de domínios específicos correto, ser de grande utilidade para um comprador. Devido à sua construção declarativa, ele se estende com facilidade a aplicações mais complexas. O principal objetivo desta seção é mostrar que alguma representação de conhecimento — em particular, a hierarquia de produto — é necessária para um agente como esse e que, uma vez que temos algum conhecimento nessa forma, o resto vem naturalmente.

12.8 RESUMO

Focalizando os detalhes de como se representa uma variedade de formas de conhecimento,

esperamos ter dado ao leitor uma ideia de como são construídas as bases de conhecimentos reais e um sentimento para as questões filosóficas interessantes que surgem. Os principais pontos são:

- A representação de conhecimento em grande escala exige uma ontologia de uso geral para organizar e reunir os vários domínios específicos do conhecimento.
- Uma ontologia de uso geral precisa cobrir ampla variedade de tipos de conhecimento e deve ser capaz, em princípio, de manipular qualquer domínio.
- A construção de uma ontologia ampla de propósito geral é um desafio significativo que ainda precisa ser plenamente realizado, apesar das estruturas atuais parecerem ser bastante robustas.
- Apresentamos uma **ontologia superior** baseada em categorias e no cálculo de eventos. Focalizamos categorias, subcategorias, partes, objetos estruturados, medidas, substâncias, eventos, tempo e espaço, mudança e crenças.
- As espécies naturais não podem ser completamente definidas na lógica, mas as suas propriedades podem ser representadas.
- Ações, eventos e tempo podem ser representados em um cálculo de situações ou em representações mais expressivas, como o cálculo de eventos. Tais representações permitem a um agente construir planos por inferência lógica.
- Apresentamos uma análise detalhada do domínio de compras da Internet, exercitando a ontologia geral e mostrando como o conhecimento do domínio pode ser utilizado por um agente de compras.
- Sistemas de representação de uso especial, como **redes semânticas** e **lógicas de descrição**, foram elaborados para ajudar na organização de uma hierarquia de categorias. A **herança** é uma forma importante de inferência, permitindo que as propriedades de objetos sejam deduzidas a partir de sua pertinência a categorias.
- A **hipótese de mundo fechado**, implementada em programas em lógica, fornece um meio simples de evitar a necessidade de especificar grande quantidade de informações negativas. É melhor interpretá-la como um **default** que pode ser anulado por informações adicionais.
- **Lógicas não monotônicas**, como **circunscrição** e **lógica default**, se destinam a captar o raciocínio default em geral.
- Os **sistemas de manutenção de verdade** manipulam atualizações e revisões do conhecimento de forma eficiente.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Briggs (1985) alega que a representação de conhecimento formal teve início com a clássica teorização indiana sobre a gramática de sânscrito (Shastric Sanskrit), que data do primeiro milênio a.C. No Ocidente, o uso de definições de termos em matemática grega antiga pode ser considerado como a primeira ocorrência: *Metafísica* de Aristóteles (literalmente, o que vem depois do livro em física) é quase um sinônimo de ontologia. Na realidade, o desenvolvimento de terminologia técnica em qualquer campo pode ser visto como uma forma de representação de conhecimento.

As primeiras discussões sobre a representação em IA tendiam a se concentrar na "representação de *problemas*", e não na "representação de *conhecimento*" [veja, por exemplo, a discussão de

Amarel (1968) do problema dos missionários e canibais]. Na década de 1970, a IA enfatizava o desenvolvimento de “sistemas especialistas” (também chamados “sistemas baseados em conhecimento”) que podiam, se fosse dado o conhecimento de domínio apropriado, equiparar ou superar o desempenho de especialistas humanos em tarefas específicas bem definidas. Por exemplo, o primeiro sistema especialista, o DENDRAL (Feigenbaum *et al*., 1971; Lindsay *et al*., 1980), interpretava a saída de um espectrômetro de massa (um tipo de instrumento usado para analisar a estrutura de compostos químicos orgânicos) com tanta precisão quanto químicos especialistas. Embora o sucesso do DENDRAL tenha ajudado a convencer a comunidade de pesquisa em IA sobre a importância da representação de conhecimento, os formalismos de representação que foram utilizados no DENDRAL são altamente específicos para o domínio da química. Com o passar do tempo, os pesquisadores ficaram interessados em formalismos e ontologias padronizadas de representação de conhecimento que podiam simplificar o processo de criação de novos sistemas especialistas. Fazendo isso, eles se aventuraram em um território antes explorado por filósofos da ciência e da linguagem. A disciplina imposta à IA pela necessidade de fazer “funcionarem” as teorias de alguém levou a um progresso mais rápido e mais profundo do que ocorreu quando esses problemas faziam parte do domínio exclusivo da filosofia (embora ele às vezes também tenha levado à repetida reinvenção da roda).

A criação de taxonomias ou classificações abrangentes remonta a tempos antigos. Aristóteles (384-322 a.C.) enfatizava fortemente esquemas de classificação e divisão em categorias. A obra *Organon*, uma coleção de trabalhos sobre lógica montada por seus alunos depois da morte do filósofo, incluiu um tratado chamado *Categorias*, em que ele tentou construir o que agora denominaríamos ontologia superior. Aristóteles também introduziu as noções de **gênero** e **espécie** para classificação de nível mais baixo. Nosso sistema atual de classificação biológica, incluindo o uso da “nomenclatura binomial” (classificação por gênero e espécie, no sentido técnico), foi criada pelo biólogo sueco Carolus Linnaeus, ou Carl von Linne (1707-1778). Os problemas associados a espécies naturais e a limites imprecisos entre categorias foram tratados por Wittgenstein (1953), Quine (1953), Lakoff (1987) e Schwartz (1977), entre outros.

O interesse em ontologias de grande escala está crescendo, como documentado pelo *Handbook on Ontologies* (Staab, 2004). O projeto OpenCyc (Lenat e Guha, 1990; Matuszek *et al*,. 2006) lançou uma ontologia com 150.000 conceitos, com uma ontologia superior semelhante à da Figura 12.1, bem como conceitos específicos como “OLED Display” e “iPhone”, que é um tipo de “telefone celular”, que por sua vez é um tipo de “eletrônica de consumo”, “telefone”, “dispositivo de comunicação sem fio” e outros conceitos. O projeto DBPedia extrai dados estruturados da Wikipedia, especificamente dos Infoboxes: as caixas de pares de atributo/valor que acompanham muitos artigos da Wikipedia (Wu e Weld, 2008;. Bizer *et al.,* 2007). Em meados de 2009, a DBPedia continha 2,6 milhões de conceitos, com cerca de 100 fatos por conceito. O grupo de trabalho IEEE P1600.1 criou a Suggested Upper Merged Ontology (SUMO) (Niles e Pease, 2001; Pease e Niles, 2002), que contém cerca de 1.000 termos de ontologia superior e links para mais de 20.000 termos de domínio específico. Stoffel *et al*. (1997) descreveram algoritmos para gerenciar eficientemente uma ontologia muito grande. Uma pesquisa de técnicas para a extração de conhecimento a partir de páginas Web é dada por Etzioni *et al*. (2008).

Na Web estão surgindo linguagens de representação. A RDF (Brickley e Guha, 2004) permite

afirmações a serem feitas na forma de triplas relacionais e fornece alguns meios para a evolução do significado de nomes ao longo do tempo. A OWL (Smith *et al*., 2004) é uma lógica de descrição que suporta inferências sobre essas triplas. Até agora, o uso parece ser inversamente proporcional à complexidade de representação: os formatos tradicionais HTML e CSS representam mais de 99% do conteúdo da Web, seguido pelos esquemas mais simples de representação, como microformatos (Khare, 2006) e RDFa (Adida e Birbeck, 2008), que utilizam marcação HTML e XHTML para adicionar atributos ao texto literal. O uso de ontologias sofisticadas RDF e OWL ainda não está generalizado, e a visão completa da Web Semântica (Berners-Lee *et al*., 2001) ainda não é extensamente utilizada. As conferências *Formal Ontology in Information Systems* (FOIS) contêm muitos artigos interessantes sobre ontologias gerais e também sobre ontologias específicas de domínios.

A taxonomia usada neste capítulo foi desenvolvida pelos autores; parte dela se baseia em nossa experiência no projeto CYC e parte no trabalho realizado por Hwang e Schubert (1993) e por Davis (1990, 2005). Uma discussão inspiradora sobre o projeto geral da representação de conhecimento de senso comum aparece na obra de Hayes (1978, 1985b) *The Naive Physics Manifesto*.

Ontologias profundas de sucesso em uma área específica incluem o projeto Gene Ontology (Consortium, 2008) e CML, a Chemical Markup Language (Murray-Rust *et al*., 2003).

Dúvidas sobre a viabilidade de uma ontologia única para *todos* os conhecimentos foram expressas por Doctorow (2001), Gruber (2004), Halevy *et al*. (2009) e Smith (2004), que afirmou: “O projeto inicial de construção de uma ontologia única (…) foi (…) em grande parte abandonado.”

O cálculo de eventos foi introduzido por Kowalski e Sergot (1986) para tratar o tempo contínuo, e houve muitas variações (Sadri e Kowalski, 1995; Shanahan, 1997) e apresentações (Shanahan, 1999; Mueller, 2006). Van Lambalgen e Hamm (2005) mostram como a lógica de eventos é mapeada dentro da linguagem que usamos para falar sobre eventos. Uma alternativa para o cálculo de eventos e de situações é o cálculo de fluentes (Thielscher, 1999). James Allen introduziu intervalos de tempo pela mesma razão (Allen, 1983, 1984), argumentando que intervalos eram muito mais naturais que situações para se raciocinar sobre eventos estendidos e concorrentes. Peter Ladkin (1986a, 1986b) introduziu intervalos de tempo “côncavos” (intervalos com lacunas; em essência, uniões de intervalos de tempo “convexos” comuns) e aplicou as técnicas da álgebra matemática abstrata à representação do tempo. Allen (1991) investiga sistematicamente a ampla variedade de técnicas disponíveis para representação do tempo; Van Beek e Manchak (1996) analisam algoritmos para raciocínio temporal.

Existem significativos pontos comuns entre a ontologia baseada em eventos dada neste capítulo e uma análise de eventos criada pelo filósofo Donald Davidson (1980). As **histórias** na ontologia de líquidos de Pat Hayes (1985a) e as **crônicas** na teoria dos planos de McDermott (1985) também constituíram influências importantes para a área e para este capítulo.

A questão do *status* ontológico de substâncias tem uma longa história. Platão afirmou que as substâncias eram entidades abstratas completamente distintas de objetos físicos; ele diria $FeitoDe(Manteiga_3, Manteiga)$ em vez de $Manteiga_3 \in Manteiga$. Isso leva a uma hierarquia de substâncias na qual, por exemplo, *ManteigaSemSal* é uma substância mais específica que *Manteiga*. A posição adotada neste capítulo, em que substâncias são categorias de objetos, foi defendida por

Richard Montague (1973). Ela também foi adotada no projeto CYC. Copeland (1993) elaborou um sério, mas não invencível, ataque. A abordagem alternativa mencionada no capítulo, em que a manteiga é um objeto que consiste em todos os objetos amanteigados do universo, foi proposta originalmente pelo lógico polonês Lésniewski (1916). Sua **mereologia** (o nome deriva da palavra grega que significa "parte") utilizava a relação parte-todo em substituição à teoria de conjuntos da matemática, com o objetivo de eliminar entidades abstratas como conjuntos. Uma exposição mais legível dessas ideias foi dada por Leonard e Goodman (1940), e o trabalho de Goodman, *The Structure of Appearance* (1977), aplica as ideias a vários problemas de representação de conhecimento. Embora alguns aspectos da abordagem mereológica sejam desajeitados — por exemplo, a necessidade de um mecanismo de herança separado, baseado em relações parte-todo — a abordagem ganhou o apoio de Quine (1960). Harry Bunt (1985) apresentou uma extensa análise de seu uso em representação de conhecimento. Casati e Varzi (1999) cobriram as partes, o todo e as localizações espaciais.

Os objetos mentais têm sido objeto de estudo intensivo em filosofia e em IA. Há três abordagens principais. A utilizada neste capítulo, baseada na lógica modal e mundos possíveis, é a abordagem clássica da filosofia (Hintikka, 1962; Kripke, 1963; Hughes e Cresswell, 1996). O livro *Reasoning about Knowledge* (Fagin *et al*., 1995) fornece uma introdução completa. A segunda abordagem é uma teoria de primeira ordem em que os objetos mentais são fluentes. Davis (2005) e Davis e Morgenstern (2005) descrevem essa abordagem. Ela se baseia no formalismo dos mundos possíveis e no trabalho de Robert Moore (1980, 1985). A terceira abordagem é uma **teoria sintática**, em que os objetos mentais são representados por cadeias de caracteres. Uma cadeia de caracteres é apenas um termo complexo que denota uma lista de símbolos, de modo que *PodeVoar*(Clark) pode ser representado pela lista de símbolos [*P*, *o*, *d*, *e*, *V*, *o*, *a*, *r*, (, *C*, *l*, *a*, *r*, *k*,)]. A teoria sintática de objetos mentais foi inicialmente estudada em profundidade por Kaplan e Montague (1960), que mostraram que levava a paradoxos se não fosse tratada com cuidado. Ernie Davis (1990) fornece uma comparação excelente das teorias sintática e modal do conhecimento.

O filósofo grego Porfírio (c. 234-305 d.C.), comentando as *Categorias* de Aristóteles, estabeleceu o que se poderia qualificar como a primeira rede semântica. Charles S. Peirce (1909) desenvolveu grafos existenciais como o primeiro formalismo de rede semântica a utilizar a lógica moderna. Ross Quillian (1961), guiado por um interesse na memória humana e no processamento de linguagens, iniciou o trabalho em redes semânticas dentro da IA. Um influente artigo de Marvin Minsky (1975) apresentou uma versão de redes semânticas chamadas **frames**; um frame era uma representação de um objeto ou categoria, com atributos e relações para outros objetos ou categorias. A questão da semântica surgiu de forma bastante intensa com relação às redes semânticas de Quillian (e as de outros que seguiram sua abordagem), com seus onipresentes e muito vagos "arcos É-UM". O famoso artigo de Woods (1975), "What's in a link?", despertou a atenção dos pesquisadores de IA para a necessidade de uma semântica precisa em formalismos de representação de conhecimento. Brachman (1979) desenvolveu esse ponto e propôs soluções. O trabalho de Patrick Hayes (1979), "The Logic of Frames", foi um corte ainda mais profundo, ao afirmar que "a maioria dos 'frames' é simplesmenteuma nova sintaxe para certas partes da lógica de primeira ordem". No ensaio de Drew McDermott (1978b), "Tarskian Semantics, or No Notation without Denotation!", o autor argumentava que a abordagem da teoria de modelos para semântica usada em lógica de primeira ordem deveria ser aplicada a todos os formalismos de representação de conhecimento. Essa continua a ser uma

ideia controversa; devemos observar que o próprio McDermott reviu sua posição em “A Critique of Pure Reason” (McDermott, 1987). Selman e Levesque (1993) discutem a complexidade da herança com exceções, mostrando que, na maioria das formulações, ela é NP-completa.

O desenvolvimento de lógicas de descrição é a fase mais recente em uma longa linha de pesquisa orientada para a descoberta de subconjuntos úteis de lógica de primeira ordem, para os quais a inferência é computacionalmente tratável. Hector Levesque e Ron Brachman (1987) mostraram que certas construções lógicas — em especial certos usos da disjunção e da negação — foram os principais responsáveis pela intratabilidade da inferência lógica. Com base no sistema KL-ONE (Schmolze e Lipkis, 1983), muitos pesquisadores desenvolveram sistemas que incorporam análise de complexidade teórica, sendo os de maior destaque o KRYPTON (Brachman *et al*., 1983) e o Classic (Borgida *et al*., 1989). O resultado foi um visível aumento na velocidade de inferência e uma compreensão muito melhor da interação entre complexidade e expressividade em sistemas de raciocínio. Calvanese *et al*. (1999) resumem o estado da arte, e Baader *et al*. (2007) apresentam um manual abrangente de lógicas de descrição. Contra essa tendência, Doyle e Patil (1991) argumentaram que a restrição da expressividade de uma linguagem torna impossível resolver certos problemas ou encoraja o usuário a evitar as restrições da linguagem por meios não lógicos.

Os três principais formalismos para lidar com a inferência não monotônica — circunscrição (McCarthy, 1980), lógica default (Reiter, 1980) e lógica não monotônica modal (McDermott e Doyle, 1980) — foram todos introduzidos em uma única edição especial do *AI Journal*. Delgrande e Schaub (2003) discutem os méritos das variantes, dados 25 anos de retrospectiva. A programação de conjuntos-resposta pode ser vista como uma extensão da negação por falha ou como um aprimoramento da circunscrição; a teoria subjacente da semântica de modelos estáveis foi introduzida por Gelfond e Lifschitz (1988), e os principais sistemas de programação de conjuntos-resposta são o DLV (Eiter *et al*., 1998) e o SMODELS (Niemelä *et. al*., 2000). O exemplo da unidade de disco vem do manual do usuário do SMODELS (Syrjänen, 2000). Lifschitz (2001) discute o uso da programação de conjuntos-resposta em planejamento. Brewka *et al*. (1997) apresentam uma boa visão geral das diversas abordagens para lógica não monotônica. Clark (1978) examina a abordagem de negação por falha para a programação em lógica e a completação de Clark. Van Emden e Kowalski (1976) mostram que todo programa Prolog sem negação tem um modelo minimal único. Nos últimos anos houve um interesse renovado em aplicações de lógicas não monotônicas a sistemas de representação de conhecimento em grande escala. Os sistemas BENINQ para manipulação de investigações de benefícios de seguros talvez tenham sido a primeira aplicação comercialmente bem-sucedida de um sistema de herança não monotônico (Morgenstern, 1998). Diversos sistemas de raciocínio não monotônicos baseados em programação em lógica estão documentados nos anais das conferências sobre *Logic Programming and Nonmonotonic Reasoning* (LPNMR).

O estudo de sistemas de manutenção de verdade começou com os sistemas TMS (Doyle, 1979) e RUP (McAllester, 1980), ambos essencialmente JTMS. Forbus e Kleer (1993) explicam em profundidade como TMSs podem ser usados em aplicações de IA. Nayak e Williams (1997) mostram como um TMS incremental eficience chamado ITMS torna possível planejar as operações de uma espaçonave da Nasa em tempo real.

Por motivos óbvios, este capítulo não poderia abordar em profundidade *todas* as áreas de

representação de conhecimento. Os três principais tópicos omitidos são:

Física qualitativa: A física qualitativa é um subcampo da representação de conhecimento que se preocupa especificamente com a construção de uma teoria lógica não numérica de objetos e processos físicos. A expressão foi cunhada por Johan de Kleer (1975), embora se possa dizer que o empreendimento teve início no BUILD de Fahlman (1974), um sofisticado planejador para construção de torres complexas de blocos. Fahlman descobriu no processo de projeto que a maior parte do esforço (em sua estimativa, 80%) se destinava à modelagem dos aspectos físicos do mundo de blocos para calcular a estabilidade de vários subconjuntos de blocos, em vez do planejamento em si. Ele esboçou um processo hipotético semelhante ao da física ingênua para explicar por que crianças pequenas podem resolver problemas como o BUILD sem acesso à aritmética de ponto flutuante em alta velocidade utilizada na modelagem física do BUILD. Hayes (1985a) utiliza "histórias" — fatias quadridimensionais de espaço-tempo semelhante aos eventos de Davidson — para construir uma física elementar bastante complexa de líquidos. Hayes foi o primeiro a provar que um banho com a banheira tampada eventualmente provocará transbordamento se a torneira continuar aberta e que uma pessoa que cair em um lago ficará completamente molhada. Davis (2008) atualiza a ontologia de líquidos que descreve o derramamento de líquidos em recipientes.

De Kleer e Brown (1985), Ken Forbus (1985) e Benjamin Kuipers (1985) independente e quase simultaneamente desenvolveram sistemas que podem raciocinar sobre o sistema físico com base em abstrações qualitativas de equações subjacentes. A física qualitativa se desenvolveu até chegar ao ponto em que se tornou possível analisar uma impressionante variedade de sistemas físicos complexos (Yip, 1991). As técnicas qualitativas foram usadas para construir projetos inovadores de relógios, limpadores de para-brisa e andadores de seis pernas (Subramanian e Wang, 1994). A coleção *Readings in Qualitative Reasoning about Physical Systems* (Weld e de Kleer, 1990), um artigo de enciclopédia por Kuipers (2001) e um artigo de handbook por Davis (2007) introduzem a área.

Raciocínio espacial: O raciocínio necessário para navegar no mundo de wumpus e no mundo de compras é trivial em comparação à rica estrutura espacial do mundo real. A primeira tentativa séria de captar o raciocínio comum sobre o espaço aparece no trabalho de Ernest Davis (1986, 1990). O cálculo de conexão de regiões de Cohn *et al*. (1997) admite uma forma de raciocínio espacial qualitativo e leva a novos tipos de sistemas de informações geográficas; consulte também Davis (2006). Como ocorre com a física qualitativa, um agente pode percorrer um longo caminho, por assim dizer, sem recorrer a uma representação métrica completa. Quando tal representação é necessária, podem ser usadas técnicas desenvolvidas em robótica (veja o Capítulo 25).

Raciocínio psicológico: O raciocínio psicológico envolve o desenvolvimento de uma *psicologia* funcional para uso por agentes artificiais no raciocínio sobre si mesmos e sobre outros agentes. Com frequência, esse raciocínio se baseia na chamada psicologia popular, que — acredita-se — os seres humanos em geral utilizam no raciocínio sobre si mesmos e sobre outros seres humanos. Quando pesquisadores de IA fornecem a seus agentes artificiais teorias psicológicas para raciocinar sobre outros agentes, as teorias frequentemente se baseiam na descrição dos pesquisadores do próprio projeto dos agentes lógicos. Atualmente, o raciocínio psicológico é mais útil no contexto da

compreensão da linguagem natural, na qual prever as intenções do falante é de grande importância.

Minker (2001) reúne artigos de pesquisadores de liderança na representação do conhecimento, resumindo 40 anos de trabalho de campo. Os anais das conferências internacionais sobre *Principles of Knowledge Representation and Reasoning* fornecem as fontes mais atualizadas de pesquisa nessa área. *Readings in Knowledge Representation* (Brachman e Levesque, 1985) e *Formal Theories of the Commonsense World* (Hobbs e Moore, 1985) são excelentes antologias sobre representação de conhecimento; a primeira se concentra mais em documentos historicamente importantes sobre linguagens de representação e formalismos, e a outra se concentra na acumulação do próprio conhecimento. Davis (1990), Stefik (1995) e Sowa (1999) apresentam introduções à representação de conhecimento de forma didática, Van Harmelen *et al.* (2007) contribuiem com um manual e uma edição especial do *AI Journal* abrange o progresso recente (Davis e Morgenstern, 2004). A conferência bienal *Theoretical Aspects of Reasoning About Knowledge* (TARK) abrange aplicações da teoria do conhecimento em IA, economia e sistemas distribuídos.

EXERCÍCIOS

12.1 Defina uma ontologia na lógica de primeira ordem para o jogo da velha. A ontologia deve conter situações, ações, quadrados, jogadores, marcações (X, O ou em branco) e a noção de ganhar, perder ou empatar o jogo. Defina também a noção de vitória forçada (ou empate): uma posição da qual um jogador pode forçar uma vitória (ou empate) com a sequência correta de ações. Escreva axiomas para o domínio. (Nota: os axiomas que enumeram quadrados diferentes e que caracterizam as posições vencedoras são bastante longos. Você não precisa escrevê-los na íntegra, mas indicar claramente com o que se parecem.)

12.2 A Figura 12.1 mostra os níveis superiores de uma hierarquia para tudo. Estenda-a para incluir tantas categorias reais quanto possível. Uma boa forma de fazer isso é considerar todos os itens da sua vida cotidiana. Isso inclui objetos e eventos. Inicie ao acordar e prossiga de maneira ordenada anotando tudo o que você vê, toca, faz e pensa. Por exemplo, uma amostra aleatória produz música, notícias, leite, caminhada, direção, gasolina, Soda Hall, tapete, conversa, Professor Fateman, frango ao curry, $7, sol, jornal diário e assim por diante.

Produza um gráfico de hierarquia simples (em uma folha grande de papel) e uma lista de objetos e categorias com as relações satisfeitas pelos membros de cada categoria. Todo objeto deve pertencer a uma categoria e toda categoria deve estar na hierarquia.

12.3 Desenvolva um sistema de representação para o raciocínio sobre janelas em uma interface de computador baseada em janelas. Em especial, a representação deve ser capaz de descrever:

- O estado de uma janela: minimizada, exibida ou inexistente.
- Qual janela (se houver) é a janela ativa.
- A posição de cada janela em determinado momento.
- A ordem (de frente para trás) das janelas sobrepostas.
- As ações de criar, destruir, redimensionar e mover janelas; alterar o estado de uma janela; e trazer uma janela para a frente. Tratar essas ações como atômicas, ou seja, não lidar com a

questão de relacioná-las com as ações do mouse. Forneça axiomas descrevendo os efeitos das ações em fluentes. Você pode usar cálculo de eventos ou cálculo de situações.

Suponha uma ontologia *contendo situações*, *ações*, *inteiros* (coordenadas para x e y) e *janelas*. Defina uma linguagem sobre essa ontologia, isto é, uma lista de constantes, símbolos de função e predicados com uma descrição de cada. Se precisar adicionar mais categorias para a ontologia (por exemplo, pixels), poderá fazê-lo, mas não se esqueça de especificá-las no seu exercício. Você pode (e deve) usar símbolos definidos no texto, mas não se esqueça de listá-los explicitamente.

12.4 Exponha o seguinte na linguagem que você desenvolveu para o exercício anterior:

a. Na situação S_0, a janela W_1 está por trás da W_2, mas está visível nas partes esquerda e direita. Não declare as coordenadas exatas para elas; descreva a situação *geral*.

b. Se uma janela for exibida, a sua borda superior é maior do que a sua borda inferior.

c. Depois de criar uma janela *w*, ela será exibida.

d. Uma janela só poderá ser minimizada se for exibida.

12.5 (Adaptada de um exemplo de Doug Lenat.) Sua missão é captar, em forma lógica, conhecimento suficiente para responder a uma série de perguntas sobre o cenário simples a seguir:

Ontem, John foi ao supermercado Safeway, de North Berkeley, e comprou dois quilos de tomates e um quilo de carne moída.

Comece tentando representar o conteúdo da sentença como uma série de asserções. Você deve escrever sentenças que tenham estrutura lógica simples (por exemplo, declarações de que os objetos têm certas propriedades, de que os objetos estão relacionados de determinada maneira, de que todos os objetos que satisfazem uma propriedade satisfazem outra). Os itens a seguir devem ajudá-lo a começar:

- Que classes, objetos e relações você precisaria ter? Quais são seus pais, irmão, e assim por diante? (Você precisará de eventos e ordenação temporal, entre outros itens.)
- Onde eles caberiam em uma hierarquia mais geral?
- Quais são as restrições e os inter-relacionamentos entre eles?
- Que detalhes você deve mostrar sobre cada um dos diversos conceitos?

Para responder às perguntas a seguir, sua base de conhecimento deve incluir conhecimento do domínio. Você terá de lidar com os tipos de itens que existem em um supermercado, o que está envolvido na compra dos itens selecionados, qual será a utilidade dos itens comprados, e assim por diante. Tente fazer sua representação tão geral quanto possível. Aqui está um exemplo trivial: não diga "As pessoas compram comida no Safeway" porque isso não o ajudará no caso daqueles que compram em outro supermercado. Também não transforme as perguntas em respostas; por exemplo, a pergunta (c) é "João comprou alguma carne?", e não "João comprou um quilo de carne moída?".

Esboce as cadeias de raciocínio que responderiam às perguntas. Se possível, use um sistema de raciocínio lógico para demonstrar a suficiência de sua base de conhecimento. Na realidade, muitos itens que você anotar talvez só estejam aproximadamente corretos, mas não se preocupe demais; a ideia é extrair o senso comum que o levará a responder a essas perguntas. Uma resposta

verdadeiramente completa para essa pergunta é *extremamente* difícil e talvez esteja além do estado da arte da representação de conhecimento atual. Porém, você deve ser capaz de reunir um conjunto consistente de axiomas para as perguntas limitadas formuladas aqui.

a. João é uma criança ou um adulto? [Adulto]

b. João agora tem pelo menos dois tomates? [Sim]

c. João comprou alguma carne? [Sim]

d. Se Maria estava comprando tomates ao mesmo tempo que João, ele a viu? [Sim]

e. Os tomates são produzidos no supermercado? [Não]

f. O que João vai fazer com os tomates? [Comê-los]

g. O Safeway vende desodorante? [Sim]

h. João trouxe algum dinheiro ou o cartão de crédito para o supermercado? [Sim]

i. João tem menos dinheiro depois de ir ao supermercado? [Sim]

12.6 Faça os acréscimos ou as alterações necessárias em sua base de conhecimento do exercício anterior, de forma que as perguntas a seguir possam ser respondidas. Inclua em seu relatório uma discussão das alterações, explicando por que elas foram necessárias, se foram secundárias ou importantes e que tipos de questões necessecitariam de outras alterações.

a. Há outras pessoas no Safeway enquanto João está lá? [Sim — os funcionários!]

b. João é vegetariano? [Não]

c. Quem é o dono do desodorante que está no Safeway? [Safeway Corporation]

d. João tinha cerca de 30 gramas de carne moída? [Sim]

e. O posto Shell vizinho ao supermercado tem gasolina? [Sim]

f. Os tomates cabem no porta-malas de João? [Sim]

12.7 Represente as sete sentenças a seguir utilizando e ampliando as representações desenvolvidas no capítulo:

a. A água é um líquido entre 0 e 100 graus.

b. A água ferve a 100 graus.

c. A água na garrafa de água de João está congelada.

d. Perrier é uma espécie de água.

e. João tem Perrier em sua garrafa de água.

f. Todos os líquidos têm um ponto de congelamento.

g. Um litro de água pesa mais que um litro de álcool.

12.8 Escreva definições para os seguintes itens:

a. *DecomposiçãoExaustivaEmPartes*

b. *PartiçãoDeParte*

c. *DisjuntoPorParte*

Essas definições devem ser análogas às definições para *DecomposiçãoExaustiva*, *Partição* e *Disjunto*. Ocorre *PartiçãoDeParte*(*s*, *GrupoDe*(*s*))? Em caso afirmativo, prove; se não, forneça um contraexemplo e defina condições suficientes sobre as quais isso é válido.

12.9 Um esquema alternativo para representar medidas envolve a aplicação da função unidades a um objeto de comprimento abstrato. Em tal esquema, seria possível escrever *Polegadas*(*Comprimento*(L_1)) = 1,5. Como esse esquema se compara com o esquema deste capítulo? As questões incluem axiomas de conversão, nomes para quantidades abstratas (como “50 dólares”) e comparações entre medidas abstratas em diferentes unidades (50 polegadas é maior que 50 centímetros).

12.10 Adicione sentenças para estender a definição do predicado *Nome*(*s*, *c*), de forma que uma cadeia como “computador laptop” seja comparada com os nomes de categorias apropriados de diversas lojas. Procure tornar sua definição geral. Teste-a examinando 10 lojas on-line e os nomes de categorias que elas fornecem para três categorias diferentes. Por exemplo, para a categoria de laptops, encontramos os nomes “Notebooks”, “Laptops”, “Computadores Notebook”, “Notebook”, “Laptops e Notebooks” e “PCs Notebooks”. Alguns desses nomes podem ser cobertos por fatos explícitos sobre *Nome*, enquanto outros podem ser cobertos por regras para tratamento de plurais, conjunções etc.

12.11 Escreva axiomas do cálculo de eventos para descrever as ações no mundo de wumpus.

12.12 Declare a relação da álgebra de intervalos que seja válida entre cada par dos seguintes eventos do mundo real:

VK: A vida do presidente Kennedy.

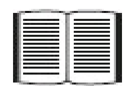
IK: A infância do presidente Kennedy.

PK: A presidência do Presidente Kennedy.

VJ: A vida do presidente Johnson.

PJ: A presidência do presidente Johnson.

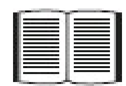
VO: A vida do presidente Obama.

 **12.13** Investigue maneiras de estender o cálculo de eventos para manipular eventos *simultâneos*. É possível evitar uma explosão combinatória de axiomas?

12.14 Construa uma representação para taxas de câmbio entre moedas que permita flutuações diárias.

12.15 Defina o predicado *Fixo*, onde *Fixo*(*Posição*(*x*)) significa que a posição do objeto *x* é fixa ao longo do tempo.

12.16 Descreva o evento de trocar algo por qualquer outra coisa. Descreva a compra como uma espécie de troca em que um dos objetos trocados é uma quantia em dinheiro.

 **12.17** Os dois exercícios anteriores pressupõem uma noção bastante primitiva de propriedade. Como exemplo, observe que o comprador começa *possuindo* as cédulas de dinheiro. Esse quadro começa a se quebrar quando, por exemplo, o dinheiro de alguém está no banco porque

não existe mais nenhuma coleção específica de cédulas que alguém possua. O quadro se complica ainda mais quando ocorrem empréstimos, arrendamentos, aluguel e custódia. Investigue os diversos conceitos comuns e legais de propriedade, e proponha um esquema pelo qual eles possam ser formalmente representados.

12.18 (Adaptada de Fagin *et al.*, 1995.) Considere um jogo com um baralho de apenas oito cartas, quatro ases e quatro reis. Distribuem-se duas cartas para cada um dos três jogadores, Alice, Bob e Carlos. Sem olhar para elas, eles colocam as cartas na testa, para que os outros jogadores possam vê-las. Em seguida, os jogadores se revezam anunciando que sabem que cartas estão na sua própria testa, assim ganhando o jogo, ou dizendo: "Eu não sei." Todo mundo sabe que os jogadores são honestos e perfeitos no raciocínio sobre as crenças.

a. Jogo 1. Alice e Bob disseram "Não sei". Carlos vê que Alice tem dois Ases (A-A) e Bob tem dois reis (K-K). O que deveria Carlos dizer? (*Dica*: Considere todos os três casos possíveis para Carlos: A-A, K-K, A-K.)

b. Descreva cada etapa do Jogo 1 usando a notação da lógica modal.

c. Jogo 2. Carlos, Alice e Bob disseram "Não sei" em seu primeiro turno. Alice tem K-K e Bob tem A-K. O que Carlos deveria dizer em seu segundo turno?

d. Jogo 3. Alice, Carlos e Bob todos dizem "Não sei" em seu primeiro turno, assim como Alice em seu segundo turno. Alice e Bob têm A-K. O que Carlos deveria dizer?

e. Prove que sempre deverá haver um ganhador para esse jogo.

12.19 A suposição da *onisciência lógica*, discutida no final da Seção 12.4, naturalmente, não é verdadeira sobre qualquer pensador real. Pelo contrário, é uma *idealização* do processo de raciocínio que pode ser mais ou menos aceitável, dependendo das aplicações. Discuta a razoabilidade do pressuposto para cada uma das seguintes aplicações de raciocínio sobre o conhecimento:

a. Jogos de conhecimento parcial com adversários, tal como os jogos de cartas. Aqui um jogador quer inferir sobre o que o seu oponente sabe sobre o estado do jogo.

b. Xadrez com um relógio. Aqui o jogador pode desejar raciocinar sobre os limites do seu oponente ou a sua própria capacidade de encontrar a melhor jogada no tempo disponível. Por exemplo, se o jogador A tem muito mais tempo do que o jogador B, então A, às vezes, faz um movimento que complica muito a situação, na esperança de obter uma vantagem porque ele tem mais tempo para elaborar a estratégia adequada.

c. Um agente de compras em um ambiente no qual existem custos de coleta de informações.

d. Raciocínio sobre criptografia de chave pública, que se baseia na intratabilidade de certos problemas computacionais.

12.20 Traduza a descrição da expressão lógica seguinte (da Seção 12.5.2) em lógica de primeira ordem e comente o resultado:

And(*Homem*, *AtLeast*(3, *Filho*), *AtMost*(2, *Filha*),
All(*Filho*, *And*(*Desempregado*, *Casado*, *All*(*Esposa*, *Médica*))),
All(*Filha*, *And*(*Professora*, *Fills*(*Departamento*, *Física*, *Matemática*))))

12.21 Lembre-se de que as informações de herança em redes semânticas podem ser captadas logicamente por sentenças de implicação adequadas. Este exercício investiga a eficiência de usar tais sentenças para herança.

a. Considere o conteúdo de informações em um catálogo de automóveis usados como o de um grande jornal americano — por exemplo, que as vans Dodge 1973 valem US$ 575,00 (ou talvez valeram algum dia). Suponha que todas essas informações (referentes a 11.000 modelos) estejam codificadas sob a forma de sentenças lógicas, como sugerimos no capítulo. Escreva três dessas sentenças, incluindo a das vans Dodge 1973. De que maneira você utilizaria as sentenças para descobrir o valor de um carro *específico*, dado um provador de teoremas de encadeamento para trás como o Prolog?

b. Compare a eficiência de tempo do método de encadeamento para trás para resolver esse problema com o método de herança utilizado em redes semânticas.

c. Explique como o encadeamento para a frente permite a um sistema baseado em lógica resolver o mesmo problema de modo eficiente, supondo-se que o jornal contenha apenas as 11.000 sentenças sobre preços.

d. Descreva uma situação em que nem o encadeamento para a frente nem o encadeamento para trás sobre as sentenças permitirão que a consulta de preço referente a um carro individual seja tratada de modo eficiente.

e. Você poderia sugerir uma solução que permita resolver esse tipo de consulta de modo eficiente em todos os casos em sistemas lógicos? [*Sugestão*: Lembre-se de que dois carros do mesmo ano e modelo têm o mesmo preço.]

12.22 Alguém poderia supor que a distinção sintática entre arcos sem retângulos e arcos com retângulos de aresta única em redes semânticas fosse desnecessária porque arcos com retângulos de aresta única sempre estão associados a categorias; um algoritmo de herança poderia simplesmente supor que um arco sem retângulo associado a uma categoria pretendesse se aplicar a todos os elementos dessa categoria. Mostre que esse argumento é uma falácia, dê exemplos de erros que surgiriam em consequência dele.

12.23 Uma parte do processo de compras que não foi abrangido nesse capítulo é a verificação da compatibilidade entre itens. Por exemplo, se for solicitada uma câmera digital, que acessórios, baterias, cartão de memória e estojos são compatíveis com a câmera? Escreva uma base de conhecimento que possa determinar a compatibilidade de um conjunto de itens e sugira substituições ou itens adicionais se o comprador fizer uma escolha que não seja compatível. A base de conhecimento deve funcionar pelo menos com uma linha de produtos e se estender facilmente para outras linhas.

12.24 Uma solução completa para o problema de correspondências inexatas com a descrição do comprador em compras é muito difícil e exige uma série completa de técnicas de processamento de linguagem natural e recuperação de informações (veja os Capítulos 22 e 23). Um pequeno passo consiste em permitir que o usuário especifique valores mínimos e máximos para diversos atributos. O comprador deve usar a gramática a seguir nas descrições de produto:

Descrição	→	*Categoria* [*Conector Modificador*] *

<table>
<tr><td>Conector</td><td>→</td><td>com” | “e” | “,”</td></tr>
<tr><td>Modificador</td><td>→</td><td>Atributo | Atributo Op Valor</td></tr>
<tr><td>Op</td><td>→</td><td>“=” | “>” | “<”</td></tr>
</table>

Aqui, *Categoria* denomina uma categoria de produtos, *Atributo* é alguma característica como “CPU” ou “preço” e *Valor* é o valor de destino para o atributo. Então, a consulta “computador com pelo menos uma CPU de 2,5 GHz e preço abaixo de US$ 500,00” deve ser expressa novamente como “computador com CPU >2,5 GHz e preço US$ 500,00”. Implemente um agente de compras que aceite descrições nessa linguagem.

12.25 Nossa descrição de compras na Internet omitiu a importante etapa de realmente *comprar* o produto. Forneça uma descrição lógica formal de comprar, usando cálculo de eventos. Isto é, defina a sequência de eventos que ocorre quando um comprador envia um pedido de compra por cartão de crédito e eventualmente paga a fatura e recebe o produto.

---

[1] Transformar uma proposição em um objeto é chamado de **reificação**, da palavra latina *res*, ou “coisa”. JohnMcCarthy propôs o termo “coisificação”, mas nunca pegou.

[2] O famoso biólogo J.B.S. Haldane deduziu “Um carinho desmedido por besouros” por parte do Criador.

[3] Os termos “evento” e “ação” podem ser utilizados de forma intercambiável. Informalmente, “ação” conota um agente, enquanto “evento” conota a possibilidade de ações sem agente.

[4] Algumas versões do cálculo de eventos não distinguem categorias de evento de instâncias das categorias.

[5] Vários sistemas antigos falharam na tentativa de distinguir entre propriedades de elementos de uma categoria e propriedades da categoria como um todo. Isso pode levar diretamente a inconsistências, como destacou Drew McDermott (1976) em seu artigo “Artificial Intelligence Meets Natural Stupidity”. Outro problema comum foi o uso de arcos *ÉUm* para relações de subconjuntos e de pertinência, em correspondência com o uso em linguagem natural: “Um gato é um mamífero” e “Fifi é um gato”. Veja o Exercício 12.22 para examinar essas questões com mais detalhes.

[6] Note que a linguagem *não* permite simplesmente declarar que um conceito ou uma categoria é um subconjunto de outro. Essa é uma política deliberada: a subsunção entre categorias deve ser derivável a partir de alguns aspectos das descrições das categorias. Se não, algo estará faltando nas descrições.

[7] CLASSIC fornece testes práticos de subordinação eficientes, mas o tempo de execução no pior caso é exponencial.

[8] Lembre-se de que a monotonicidade exige que todas as sentenças que são consequências lógicas permaneçam consequências lógicas depois que novas sentenças forem adicionadas à BC. Isto é, se $BC \models \alpha$ então $BC \land \beta \models \alpha$.

[9] No caso da hipótese de mundo fechado, um modelo é preferível em relação a outro se ele tem um número menor de átomos verdadeiros, isto é, os modelos preferidos são modelos **minimais**. Existe uma conexão natural entre a Hipótese de Mundo Fechado (HMF) e BCs de cláusulas definidas porque o ponto fixo alcançado pelo encadeamento para a frente em tais BCs é o único modelo minimal. Veja a Seção 7.5.4 para mais detalhes.

[10] A revisão de crenças frequentemente é comparada com a **atualização de crenças**, que ocorre quando uma base de conhecimento é revisada para refletir uma mudança no mundo, em vez de novas informações sobre um mundo fixo. A atualização de crenças combina a revisão de crenças com o raciocínio sobre o tempo e a mudança; ela também está relacionada ao processo de **filtragem** descrito no Capítulo 15.

[11] Uma alternativa para a estratégia de seguir links é usar um mecanismo de pesquisa na Internet; a tecnologia por trás da pesquisa na Internet, chamada recuperação de informações, será examinada na Seção 22.3.

PARTE IV

Conhecimento incerto e pensamento

CAPÍTULO 13

Quantificando a incerteza

Em que vemos como um agente pode domar a incerteza com graus de crença.

13.1 COMO AGIR EM MEIO À INCERTEZA

Os agentes podem precisar lidar com a **incerteza**, seja devido à observabilidade parcial, ao não determinismo ou a uma combinação dos dois. Um agente pode não saber ao certo em que estado está ou onde terminará após uma sequência de ações.

Vimos agentes de resolução de problemas (Capítulo 4) e agentes lógicos (Capítulos 7 e 11) projetados para lidar com a incerteza, mantendo o controle de um **estado de crença** — uma representação do conjunto de todos os estados possíveis do mundo em que possam estar — e gerando um plano de contingência que trate de qualquer eventualidade possível que seus sensores possam relatar durante a execução. Entretanto, apesar de suas muitas virtudes, essa abordagem tem desvantagens significativas quando tomada literalmente como uma receita para a criação de programas do agente:

- Ao interpretar a informação parcial do sensor, um agente lógico deve considerar cada explanação *logicamente possível* das observações, não importa o quão improvável seja. Isso leva a representações de estados de crença impossivelmente grandes e complexos.
- Um plano de contingência correto que lida com toda eventualidade pode crescer arbitrariamente e deve considerar as contingências arbitrariamente improváveis.
- Às vezes, não há um plano garantido de alcançar o objetivo — mesmo assim o agente deve agir. Deve ter alguma maneira de comparar os méritos dos planos que não são garantidos.

Suponha, por exemplo, que um táxi automatizado tenha o objetivo de entregar um passageiro no aeroporto a tempo. O agente faz um plano, A_{90}, que envolve sair de casa 90 minutos antes da partida do voo e dirigir a uma velocidade razoável. Mesmo que o aeroporto seja apenas a oito quilômetros de distância, o agente lógico do táxi não será capaz de concluir com certeza se o "Plano A_{90} vai conduzir ao aeroporto a tempo". Em vez disso, chegará à conclusão mais fraca de que o "Plano A_{90} vai conduzir ao aeroporto a tempo se o carro não quebrar ou ficar sem combustível, e não se envolver em um acidente, e não houver acidentes na ponte e o avião não decolar mais cedo, e nenhum

meteorito bater no carro, e…” Nenhuma dessas condições pode ser deduzida com certeza, assim o sucesso do plano não pode ser inferido. Esse é o **problema de qualificação**, para o qual até agora não vimos nenhuma solução real.

Todavia, vamos supor que A_{90} *seja* de fato a alternativa correta. O que queremos dizer com isso? Como vimos no Capítulo 2, queremos dizer que, de todos os planos que poderiam ser executados, espera-se que o plano A_{90} maximize a medida de desempenho do agente (onde a expectativa é relativa ao conhecimento do agente sobre o ambiente). A medida de desempenho inclui chegar ao aeroporto a tempo para o voo, evitando uma longa e improdutiva espera no aeroporto, e evitando multas por excesso de velocidade ao longo do caminho. As informações que o agente tem não podem garantir quaisquer desses resultados para A_{90}, mas podem fornecer algum grau de crença de que os resultados serão alcançados. Outros planos, como A_{180}, poderiam aumentar a crença do agente de que ele chegará ao aeroporto a tempo, mas também aumentarão a probabilidade de uma longa espera. *Então, a alternativa correta — a **decisão racional** — depende tanto da importância relativa de várias metas quanto da probabilidade de que elas serão alcançadas e em que grau*. O restante desta seção apura essas ideias como preparação para o desenvolvimento das teorias gerais de raciocínio incerto e decisões racionais que apresentaremos neste capítulo e em capítulos subsequentes.

13.1.1 Resumindo a incerteza

Vamos considerar um exemplo de raciocínio incerto: o diagnóstico de dor de dente de um paciente. Diagnóstico — seja em medicina, conserto de automóveis ou qualquer outra atividade — é uma tarefa que quase sempre envolve a incerteza. Vamos tentar definir regras para diagnóstico odontológico utilizando a lógica proposicional, de forma que possamos ver como a abordagem lógica se desenvolve. Considere a regra simples a seguir:

$$DorDeDente \Rightarrow Cárie.$$

O problema é que essa regra está errada. Nem todos os pacientes com dores de dentes têm cáries; alguns deles têm gengivite, abscessos ou algum dentre vários outros problemas:

$$DorDeDente \Rightarrow Cárie \lor Gengivite \lor Abscessos \ldots$$

Infelizmente, a fim de tornar a regra verdadeira, temos de adicionar uma lista quase ilimitada de causas possíveis. Poderíamos tentar transformar a regra em uma regra causal:

$$Cárie \Rightarrow DorDeDente.$$

No entanto, essa regra também não é correta; nem todas as cáries causam dor. O único modo de corrigir a regra é torná-la logicamente exaustiva: aumentar o lado esquerdo com todas as qualificações exigidas para que uma cárie cause dor de dente. Tentar usar a lógica de primeira ordem

para lidar com um domínio como diagnóstico médico é uma abordagem falha, por três razões principais:

- **Preguiça:** É trabalhoso demais listar o conjunto completo de antecedentes ou consequentes necessários para assegurar uma regra sem exceções, e é muito difícil usar tais regras.
- **Ignorância teórica:** A ciência médica não tem nenhuma teoria completa para o domínio.
- **Ignorância prática:** Ainda que todas as regras sejam conhecidas, poderíamos estar inseguros quanto a um paciente específico porque nem todos os testes necessários foram ou podem ser executados.

☞ A conexão entre dores de dentes e cáries não é apenas uma consequência lógica, em um sentido ou no outro. Isso é típico do domínio médico, bem como da maioria dos outros domínios de julgamento: jurídicos, de negócios, de projeto, de consertos de automóveis, de jardinagem, de encontros, e assim por diante. O conhecimento do agente pode, na melhor das hipóteses, fornecer apenas um **grau de crença** nas sentenças relevantes. Nossa principal ferramenta para lidar com graus de crença será a **teoria da probabilidade**. Na terminologia da Seção 8.1, os **compromissos ontológicos** da teoria da lógica e da probabilidade são os mesmos — que o mundo é composto de fatos que são ou não válidos em qualquer caso particular —, mas os **compromissos epistemológicos** são diferentes: um agente lógico acredita que cada sentença seja verdadeira ou falsa ou não tem opinião, enquanto um agente probabilístico pode ter um grau de crença numérico entre 0 (para sentenças que são certamente falsas) e 1 (certamente verdadeiras).

A probabilidade proporciona um meio para **resumir** *a incerteza que vem de nossa preguiça e ignorância*, resolvendo assim o problema de qualificação. Talvez não saibamos com certeza o que aflige determinado paciente, mas acreditamos que exista, digamos, uma chance de 80% — isto é, uma probabilidade igual a 0,8 — de que o paciente tenha uma cárie, caso ele esteja sentindo dor de dente. Ou seja, esperamos que, de todas as situações indistinguíveis da situação atual, até onde vai o nosso conhecimento, o paciente terá uma cárie em 80% delas. Essa crença poderia ser derivada de dados estatísticos — 80% dos pacientes com dor de dente vistos até agora tinham cáries — ou de algumas regras gerais, ou ainda de uma combinação de fontes de evidências.

Um ponto confuso é que, no momento do nosso diagnóstico, não há incerteza no mundo real: o paciente tem uma cárie ou não. Então, o que significa dizer que a probabilidade de uma cárie é de 0,8? Não deveria ser 0 ou 1? A resposta é que as declarações de probabilidade são feitas com respeito a um estado de conhecimento, não com relação ao mundo real. Dizemos: “A probabilidade de a paciente ter uma cárie, uma *vez que tem dor de dente*, é de 0,8.” Se soubermos depois que o paciente tem um histórico de doenças da gengiva, podemos fazer uma declaração diferente: “A probabilidade de que o paciente tenha uma cárie, uma vez que tem uma dor de dente e um histórico de doença da gengiva, é de 0,4.” Se reunirmos evidências mais conclusivas contra a cárie, podemos dizer: “A probabilidade de que a paciente tenha uma cárie, dado tudo o que sabemos agora, é quase 0.” Observe que essas declarações não se contradizem mutuamente, cada uma é uma afirmação separada sobre um diferente estado do conhecimento.

13.1.2 Incerteza e decisões racionais

Considere novamente o plano A_{90} para chegar ao aeroporto. Suponha que ele apresente uma chance de 97% de pegar o voo. Isso significa que é uma escolha racional? Não necessariamente: pode haver outros planos, como o A_{180}, com maiores probabilidades. Se for vital não perder o voo, valerá a pena se arriscar a uma espera mais longa no aeroporto. O que dizer do plano A_{1440}, que envolve sair de casa com 24 horas de antecedência? Na maioria das circunstâncias, essa não é uma boa escolha porque, embora garanta a chegada a tempo, envolve uma espera intolerável — sem mencionar a possibilidade pouco agradável da comida do aeroporto.

Para fazer tais escolhas, primeiro um agente deve ter **preferências** entre os diferentes **resultados** dos vários planos. Um resultado específico é um estado completamente especificado, incluindo fatores como o agente chegar a tempo ou não e a duração da espera no aeroporto. Empregaremos a **teoria da utilidade** para representar e raciocinar com preferências. A teoria da utilidade diz que todo estado tem determinado grau de utilidade (ou seja, ele tem certa utilidade) para um agente e que o agente preferirá estados com utilidade mais alta.

A utilidade de um estado é relativa ao agente. Por exemplo, a utilidade de um estado em uma partida de xadrez em que a peça branca colocou a preta em xeque é obviamente alta para o agente que joga com a branca, mas baixa para o agente que joga com a preta. Mas não podemos ir estritamente pelo número de pontos 1, 1/2 e 0 ditados pelas regras do torneio de xadrez — alguns jogadores (incluindo os autores) podem ficar excitados em empatar com o campeão mundial de xadrez, enquanto outros jogadores (incluindo o campeão mundial anterior) talvez não tenham tanto prazer. Não existe nenhuma maneira de medir o gosto ou as preferências: você pode pensar que um agente que prefere sorvete de açaí a biscoito de chocolate é estranho ou mesmo mal orientado, mas não se pode dizer que o agente é irracional. Uma função utilidade pode contar com qualquer conjunto de preferências — peculiar ou típico, nobre ou perverso. Observe que as utilidades podem levar em conta o altruísmo simplesmente incluindo o bem-estar de outras pessoas como um dos fatores.

Preferências, sendo expressas por utilidades, são combinadas com as probabilidades na teoria geral de decisões racionais chamada **teoria da decisão**:

Teoria da decisão = teoria da probabilidade + teoria da utilidade.

A ideia fundamental da teoria da decisão é que *um agente é racional se e somente se escolhe a ação que resulta na mais alta utilidade esperada*, *calculada como a média sobre todos os resultados possíveis da ação*. Isso é chamado princípio de **utilidade máxima esperada** (UME). Observe que “esperada” pode parecer um termo vago e hipotético, mas como é utilizado aqui tem um significado preciso: significa a “média” ou “média estatística” dos resultados ponderada pela probabilidade do resultado. Vimos esse princípio em ação no Capítulo 5 quando focalizamos de forma resumida decisões ótimas em partidas de gamão; é de fato um princípio completamente geral.

A Figura 13.1 esboça a estrutura de um agente que usa a teoria da decisão para selecionar ações. O agente é idêntico, em um nível abstrato, ao agente lógico descrito nos Capítulos 4 e 7 que mantém um estado de crença refletindo a história da percepção atual. A principal diferença é que a decisão teórica do estado de crença do agente não representa apenas as *possibilidades* dos estados do

mundo, mas também suas *probabilidades*. Dado o estado de crença, o agente pode fazer prognósticos probabilísticos de resultados de ações e, consequentemente, selecionar a ação com a mais alta utilidade esperada. Este capítulo e o próximo se concentram na tarefa de representar e efetuar cálculos com informações probabilísticas em geral. O Capítulo 15 lida com métodos correspondentes às tarefas específicas de representar e atualizar o estado de crença e de prognosticar o ambiente. O Capítulo 16 aborda a teoria da utilidade em maior profundidade, e o Capítulo 17 desenvolve algoritmos para as sequências de planejamento de ações em ambientes incertos.

função AGENTE-TD(*percepção*) **retorna** uma *ação*
  variáveis estáticas: *estado_de_crença*, crenças probabilísticas sobre o estado atual do mundo
    ação, a ação do agente

  atualizar *estado_de_crença* com base em *ação* e *percepção*
  calcular probabilidades de resultados de ações,
    dadas descrições de ações e o *estado_de_crença* atual
  selecionar *ação* com utilidade esperada mais alta
    dadas as probabilidades de resultados e informações de utilidade
  retornar *ação*

Figura 13.1 Um agente de teoria da decisão que seleciona ações racionais.

13.2 NOTAÇÃO BÁSICA DE PROBABILIDADE

Para que o nosso agente represente e utilize a informação probabilística, precisamos de uma linguagem formal. A linguagem da teoria da probabilidade tem sido tradicionalmente informal, escrita por matemáticos humanos a outros matemáticos humanos. O Apêndice A inclui uma introdução-padrão para a teoria da probabilidade elementar; aqui, tomamos uma abordagem mais adequada às necessidades da IA e mais consistente com os conceitos de lógica formal.

13.2.1 Sobre o que versam as probabilidades

Como as afirmações lógicas, as afirmações probabilísticas são acerca de mundos possíveis. Considerando que as afirmações lógicas dizem que os mundos possíveis são estritamente descartáveis (todos aqueles em que a afirmação é falsa), as afirmações probabilísticas versam sobre o quão prováveis são os vários mundos. Na teoria da probabilidade, o conjunto de todos os mundos possíveis é chamado de **espaço amostral**. Os mundos possíveis são *mutuamente exclusivos e exaustivos* — dois mundos possíveis não podem coexistir, e um mundo possível deve ser sempre válido. Por exemplo, se jogamos dois dados (distintos), existem 36 mundos possíveis a considerar: (1,1), (1,2),…, (6,6). A letra grega Ω (ômega maiúsculo) é usada para se referir ao espaço amostral, e ω (ômega minúsculo) refere-se aos elementos do espaço, isto é, aos mundos possíveis particulares.

Um **modelo de probabilidade** totalmente especificado associa uma probabilidade numérica $P(\omega)$ a cada mundo possível.[1] Os axiomas básicos da teoria da probabilidade dizem que todo mundo possível tem uma probabilidade entre 0 e 1 e que a probabilidade total do conjunto de mundos possíveis é 1:

$$0 \leq P(\omega) \leq 1 \text{ para cada } \omega \text{ e } \sum_{\omega \in \Omega} P(\omega) = 1 . \qquad (13.1)$$

Por exemplo, se assumirmos que os dois dados não são "viciados" e um lançamento não interfere no outro, cada um dos mundos possíveis (1,1), (1,2),…, (6,6) tem probabilidade 1/36. Por outro lado, se os dados conspirarem para produzir o mesmo número, os mundos (1,1), (2,2), (3,3) etc. poderão ter probabilidades maiores, deixando os outros com probabilidades menores.

Afirmações probabilísticas e consultas geralmente não são sobre mundos possíveis particulares, mas sobre os seus conjuntos. Por exemplo, poderíamos estar interessados nos casos em que os dois dados chegam ao resultado 11, os casos em que são jogados em duplas, e assim por diante. Em teoria da probabilidade, esses conjuntos são chamados **eventos** — um termo já usado extensivamente no Capítulo 12 para um conceito diferente. Em IA, os conjuntos são sempre descritos por **proposições** em uma linguagem formal (tal linguagem será descrita na Seção 13.2.2). Para cada proposição, o conjunto correspondente contém apenas aqueles mundos possíveis onde a proposição é válida. A probabilidade associada a uma proposição é definida como sendo a soma das probabilidades dos mundos nos quais é válida:

$$\text{Para qualquer proposição } \phi, P(\phi) = \sum_{\omega \in \phi} P(\omega) . \qquad (13.2)$$

Por exemplo, ao jogar dados que não são viciados, temos $P(Total = 11) = P((5, 6)) + P((6, 5)) = 1/36 + 1/36 = 1/18$. Observe que a teoria da probabilidade não requer conhecimento completo das probabilidades de cada mundo possível. Por exemplo, se acreditamos que os dados conspiram para produzir o mesmo número, podemos *afirmar* que $P(duplas) = 1/4$ sem saber se os dados preferem a dupla de 6 à de 2. Tal como aconteceu com asserções lógicas, essa asserção *restringe* o modelo probabilístico subjacente sem determiná-lo totalmente.

Probabilidades, tais como $P(Total = 11)$ e $P(duplas)$ são chamadas **probabilidades incondicionais** ou **anteriores** (e, às vezes, abreviado apenas como "anteriores"); elas se referem a graus de crença em proposições *na ausência de qualquer outra informação*. Na maioria das vezes, no entanto, temos alguma informação, geralmente chamada **evidência**, que já foi revelada. Por exemplo, o primeiro dado pode já estar mostrando um 5 e estamos esperando ansiosamente que o outro pare de girar. Nesse caso, não estamos interessados na probabilidade incondicional do lançamento em duplas, mas na probabilidade **condicional** ou **posterior** (ou apenas "posterior") de lançamento em duplas *considerando que o valor do primeiro dado é 5*. Essa probabilidade é escrita como $P(duplas \mid Dado_1 = 5)$, onde o "|" é pronunciado como "considerando que". Da mesma forma, se eu estou indo ao dentista para um *check-up* regular, a probabilidade $P(cárie) = 0{,}2$ pode ser interessante, mas se estou indo ao dentista porque tenho uma dor de dente, é $P(cárie \mid dor\ de\ dente) = 0{,}6$ que importa. Observe que a precedência de "|" é tal que qualquer expressão da forma $P(\ldots|\ldots)$ sempre significa $P((\ldots) \mid (\ldots))$.

É importante compreender que $P(cárie) = 0{,}2$ ainda é *válido* após a *dor de dente* ter sido observada; ela simplesmente não é especialmente útil. Ao tomar decisões, um agente precisa estipular sobre *todas* as evidências que tem observado. Também é importante entender a diferença entre condicionar e implicação lógica. A afirmação de que $P(cárie \mid dor\ de\ dente) = 0{,}6$ não significa que "sempre que *dor de dente* for verdadeiro, concluir que *cárie* é verdadeiro com probabilidade 0,6", em vez disso significa: "Sempre que *dor de dente* for verdadeiro *e não temos mais informações*, concluir que *cárie* é verdadeiro com probabilidade 0,6." A condição extra é importante; por exemplo, se tivéssemos a informação adicional de que o dentista não encontrou cáries, definitivamente não desejaríamos concluir que *cárie* é verdadeiro com probabilidade 0,6; em vez disso, precisaríamos utilizar $P(cárie \mid dor\ de\ dente \land \neg cárie) = 0$.

Matematicamente falando, as probabilidades condicionais são definidas em termos de probabilidades incondicionais como segue: para quaisquer proposições *a* e *b*, temos

$$P(a \mid b) = \frac{P(a \land b)}{P(b)}, \tag{13.3}$$

que é válido sempre que $P(b) > 0$. Por exemplo,

$$P(dupla \mid Dado_1 = 5) = \frac{P(dupla \land Dado_1 = 5)}{P(Dado_1 = 5)}.$$

A definição faz sentido se você lembrar de observar que *b* descarta todos os mundos possíveis onde *b* é falso, deixando um conjunto cuja probabilidade total é apenas $P(b)$. Dentro desse conjunto, o *a*-mundo satisfaz $a \land b$ e constitui uma fração de $P(a \land b)/P(b)$.

A definição de probabilidade condicional, Equação 13.3, pode ser escrita de uma forma diferente chamada de **regra do produto**:

$$P(a \land b) = P(a \mid b)\, P(b).$$

A regra do produto é talvez mais fácil de lembrar: ela vem do fato de que, para *a* e *b* ser verdadeiro, é necessário que *a* seja verdadeiro, dado *b*.

13.2.2 A linguagem das proposições em afirmações de probabilidade

Neste capítulo e no próximo, as proposições que descrevem conjuntos de mundos possíveis são escritas com uma notação que combina elementos de lógica proposicional e notação de satisfação de restrição. A terminologia da Seção 2.4.7 é uma **representação fatorada**, em que o mundo possível é representado por um conjunto de variável/pares de valor.

As variáveis, na teoria da probabilidade, são chamadas de **variáveis aleatórias** e seus nomes começam com letra maiúscula. Assim, no exemplo do dado, *Total* e $Dado_1$ são variáveis aleatórias. Cada variável aleatória tem um **domínio** — o conjunto de valores possíveis que pode assumir. O domínio de *Total* para dois dados é o conjunto {2,…, 12} e o domínio de $Dado_1$ é {1,…, 6}. Uma variável aleatória booleana tem o domínio {*verdadeiro*, *falso*} (note que os valores estão sempre

com letras minúsculas); por exemplo, a proposição de que as duplas são lançadas pode ser escrita como *Duplas* = *verdadeiro*. Por convenção, as proposições da forma *A* = *verdadeiro* são abreviadas simplesmente como *a*, enquanto *A* = *falso* é abreviado como ¬*a* (*duplas*, *cárie* e *dor de dente* na seção anterior são abreviaturas desse tipo). Como em PSR, os domínios podem ser conjuntos de símbolos arbitrários; poderíamos escolher o domínio de *Idade* como {*juvenil*, *adolescente*, *adulto*} e o domínio de *Tempo* como {*ensolarado*, *chuvoso*, *nublado*, nevando}. Quando nenhuma ambiguidade é possível, é comum usar um valor por si só para representar a proposição de que determinada variável tem aquele valor; assim, ensolarado pode representar *Tempo* = *ensolarado*.

Os exemplos anteriores têm domínios finitos. As variáveis podem ter domínios infinitos também — discretos (como os inteiros) ou contínuos (como os reais). Para qualquer variável com domínio ordenado, as desigualdades também são permitidas, como *NúmeroDeAtomosNoUniverso* ≥ 10^{70}.

Finalmente, podemos combinar esses tipos de proposições elementares (incluindo as formas abreviadas das variáveis booleanas) usando os conectivos da lógica proposicional. Por exemplo, podemos expressar que "a probabilidade que o paciente tenha uma cárie, uma vez que é um adolescente, sem dor de dente, é de 0,1" como segue:

$$P\,(\textit{cárie} \mid \neg\, \textit{dor de dente} \wedge \textit{adolescente}) = 0{,}1.$$

Às vezes vamos desejar falar sobre as probabilidades de *todos* os valores possíveis de uma variável aleatória. Podemos escrever:

$$P\,(\textit{Tempo} = \textit{ensolarado}) = 0{,}6$$
$$P\,(\textit{Tempo} = \textit{chuvoso}) = 0{,}1$$
$$P\,(\textit{Tempo} = \textit{nublado}) = 0{,}29$$
$$P\,(\textit{Tempo} = \textit{neve}) = 0{,}01,$$

mas, abreviando, teremos

$$\mathbf{P}(\textit{Tempo}) = \langle 0{,}6;\ 0{,}1;\ 0{,}29;\ 0{,}01 \rangle,$$

onde o **P** em negrito indica que o resultado é um vetor de números e onde assumimos uma ordenação predefinida ⟨*ensolarado*, *chuvoso*, *nublado*, nevando⟩ no domínio do *Tempo*. Dizemos que a declaração **P** define uma **distribuição de probabilidade** para a variável aleatória *Tempo*. A notação **P** também é utilizada para distribuições condicionais: $\mathbf{P}(X \mid Y)$, que dá os valores de $\mathbf{P}(X = x_i \mid Y = y_j)$ para cada par possível *i*, *j*.

Para variáveis contínuas, não é possível escrever toda a distribuição como um vetor porque há um número infinito de valores. Em vez disso, podemos definir a probabilidade de que uma variável aleatória assume algum valor de *x* como uma função parametrizada de *x*. Por exemplo, a sentença

$$P\,(\textit{TempMeioDia} = x) = \textit{Uniforme}_{[18C,\, 26C]}(x)$$

expressa a crença de que a temperatura ao meio-dia é distribuída uniformemente entre 18-26 graus Celsius. Chamamos isso de **função densidade de probabilidade**.

Funções densidade de probabilidade (às vezes chamadas de **fdps**) diferem em significado de

distribuições discretas. Dizer que a densidade de probabilidade é uniforme a partir de 18*C* até 26*C* significa que há uma chance de 100% de que a temperatura vai cair em algum lugar naquela região com amplitude 8*C* e 50% de chance de cair em qualquer região com amplitude 4*C*, e assim por diante. Escrevemos a probabilidade de densidade de uma variável aleatória contínua *X* ao valor *x* como $P(X = x)$ ou simplesmente $P(x)$; a definição intuitiva de $P(x)$ é a probabilidade de que *X* cai dentro de uma pequena região arbitrariamente iniciando em *x*, dividido pela largura da região:

$$P(x) = \lim_{dx \to 0} P(x \leq X \leq x + dx)/dx \ .$$

Para *TempMeioDia* temos

$$P(\mathit{TempMeioDia} = x) = \mathit{Uniforme}_{[18C,26C]}(x) = \begin{cases} \frac{1}{8C} \text{ if } 18C \leq x \leq 26C \\ 0 \text{ caso contrário} \end{cases},$$

onde *C* representa centígrados. Em $P(\mathit{TempMeioDia}= 20{,}18C) = \frac{1}{8C}$, observe que $\frac{1}{8C}$ não é uma probabilidade, é uma densidade de probabilidade. A probabilidade de que *TempMeioDia* seja exatamente 20,18*C* é zero porque 20,18*C* é uma região de largura 0. Alguns autores utilizam símbolos diferentes para distribuições discretas e funções de densidade; nós utilizamos *P* em ambos os casos, uma vez que raramente surge confusão e as equações são geralmente idênticas. Observe que as probabilidades são números que não têm unidade, enquanto as funções de densidade são medidas com uma unidade, nesse caso graus recíprocos.

Além de distribuições sobre variáveis simples, precisamos de uma notação para distribuições sobre variáveis múltiplas. Para isso é utilizado a vírgula. Por exemplo, **P** (*Tempo*, *Cárie*) indica as probabilidades de todas as combinações de valores de *Tempo* e de *Cárie*. Essa é uma tabela de probabilidades 4 × 2 chamada de **distribuição de probabilidade conjunta** de *Tempo* e de *Cárie*. Podemos também misturar as variáveis com e sem valores; **P** (*ensolarado*, *Cárie*) seria um vetor de dois elementos dando as probabilidades de um dia ensolarado com cárie e um dia ensolarado sem cárie. A notação **P** torna certas expressões muito mais concisas do que poderiam ser. Por exemplo, as regras dos produtos para todos os valores possíveis de *Tempo* e *Cárie* podem ser escritas como uma equação única:

P (*Tempo, Cárie*) = **P** (Tempo | *Cárie*) **P** (*Cárie*),

em vez das 4 × 2 = 8 equações (utilizando abreviaturas *W* e *C*):

$$\begin{aligned}
&P(W = \mathit{ensolarado} \land C = \mathit{verdadeiro}) = P(W = \mathit{ensolarado} \mid C = \mathit{verdadeiro})\, P(C = \mathit{verdadeiro}) \\
&P(W = \mathit{chuvoso} \land C = \mathit{verdadeiro}) = P(W = \mathit{chuvoso} \mid C = \mathit{verdadeiro})\, P(C = \mathit{verdadeiro}) \\
&P(W = \mathit{nublado} \land C = \mathit{verdadeiro}) = P(W = \mathit{nublado} \mid C = \mathit{verdadeiro})\, P(C = \mathit{verdadeiro}) \\
&P(W = \mathit{neve} \land C = \mathit{verdadeiro}) = P(W = \mathit{neve} \mid C = \mathit{verdadeiro})\, P(C = \mathit{verdadeiro}) \\
&P(W = \mathit{ensolarado} \land C = \mathit{falso}) = P(W = \mathit{ensolarado} \mid C = \mathit{falso})\, P(C = \mathit{falso}) \\
&P(W = \mathit{chuvoso} \land C = \mathit{falso}) = P(W = \mathit{chuvoso} \mid C = \mathit{falso})\, P(C = \mathit{falso}) \\
&P(W = \mathit{nublado} \land C = \mathit{falso}) = P(W = \mathit{nublado} \mid C = \mathit{falso})\, P(C = \mathit{falso}) \\
&P(W = \mathit{neve} \land C = \mathit{falso}) = P(W = \mathit{neve} \mid C = \mathit{falso})\, P(C = \mathit{falso}).
\end{aligned}$$

Como um caso degenerado, **P**(*ensolarado*, *cárie*) não tem variáveis e, portanto, é um vetor de um elemento que é a probabilidade de um dia ensolarado com uma cárie, que também poderia ser escrito

como **P**(*ensolarado*, *cárie*) ou **P**(*cárie* ∧ *ensolarado*). Utilizaremos por vezes a notação **P** para obter resultados sobre valores individuais de *P* e, quando dizemos que "**P**(*ensolarado*) = 0,6" é realmente uma abreviatura para "**P**(*ensolarado*), é o vetor de um elemento <0,6>, que significa que **P**(*ensolarado*) = 0,6".

Agora definimos uma sintaxe para proposições e afirmações de probabilidade e temos parte da semântica dada: a Equação 13.2 define a probabilidade de uma proposição como a soma das probabilidades de mundos nos quais é válida. Para completar a semântica, é preciso dizer quais são os mundos e como determinar se uma proposição é válida no mundo. Tomamos emprestada essa parte diretamente da semântica da lógica proposicional, como segue. *Um mundo possível é definido para ser uma atribuição de valores a todas as variáveis aleatórias consideradas.* É fácil verificar que essa definição satisfaz o requisito básico de que os mundos possíveis são mutuamente exclusivos e exaustivos (Exercício 13.5). Por exemplo, se as variáveis aleatórias são *Cárie*, *Dor de dente* e *Tempo*, então existem 2 × 2 × 4 = 16 mundos possíveis. Além disso, a verdade de qualquer proposição dada, não importa o quanto seja complexa, pode ser facilmente determinada em tais mundos utilizando a mesma definição recursiva de verdade, como de fórmulas em lógica proposicional.

A partir da definição anterior dos mundos possíveis, segue que um modelo de probabilidade é completamente determinado pela distribuição conjunta de todas as variáveis aleatórias — a chamada **distribuição de probabilidade conjunta completa**. Por exemplo, se as variáveis são *Dor de dente*, *Cárie* e *Tempo*, a distribuição conjunta completa é dada por **P**(*Cárie*, *Dor de dente*, *Tempo*). Essa distribuição conjunta pode ser representada como uma tabela 2 × 2 × 4 com 16 entradas. Como cada probabilidade da proposição é a soma dos mundos possíveis, uma distribuição conjunta completa é, em princípio, suficiente para calcular a probabilidade de qualquer proposição.

13.2.3 Axiomas de probabilidade e sua razoabilidade

Os axiomas básicos da probabilidade (Equações 13.1 e 13.2) implicam certas relações entre os graus de crença que podem ser atribuídos às proposições logicamente relacionadas. Por exemplo, podemos derivar a relação familiar entre a probabilidade de uma proposição e a probabilidade de sua negação:

$$\begin{aligned} P(\neg a) &= \textstyle\sum_{\omega \in \neg a} P(\omega) && \text{pela Equação 13.2} \\ &= \textstyle\sum_{\omega \in \neg a} P(\omega) + \sum_{\omega \in a} P(\omega) - \sum_{\omega \in a} P(\omega) && \\ &= \textstyle\sum_{\omega \in \Omega} P(\omega) - \sum_{\omega \in a} P(\omega) && \text{agrupando os dois primeiros termos} \\ &= 1 - P(a) && \text{por 13.1 e 13.2.} \end{aligned}$$

Podemos também derivar a bem conhecida fórmula da probabilidade de uma disjunção, às vezes chamada de **princípio de inclusão-exclusão**:

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b). \qquad (13.4)$$

Essa regra é facilmente lembrada observando que os casos em que *a* é válido, junto com os casos em que *b* é válido, certamente envolvem todos os casos em que *a* ∨ *b* é válido, mas, somando os

dois conjuntos de casos, conta sua interseção duas vezes, por isso precisamos subtrair $P(a \land b)$. A demonstração é deixada como exercício (Exercício 13.6).

As Equações 13.1 e 13.4 são frequentemente chamadas de **axiomas de Kolmogorov**, em honra ao matemático russo Andrei Kolmogorov, que mostrou como construir o restante da teoria da probabilidade a partir desse fundamento simples e como lidar com as dificuldades causadas pelas variáveis contínuas.[2] Enquanto a Equação 13.2 tem um sabor de definição, a Equação 13.4 revela que os axiomas realmente restringem os graus de crença que um agente pode ter sobre as proposições logicamente relacionadas. Isso é análogo ao fato de que um agente lógico não pode acreditar simultaneamente em A, B e $\neg(A \land B)$ porque não existe um mundo possível no qual todos os três sejam verdadeiros. Com probabilidades, no entanto, as declarações não se referem ao mundo diretamente, mas ao próprio estado de conhecimento do agente. Por que, então, um agente não pode manter o seguinte conjunto de crenças (mesmo que elas violem os axiomas de Kolmogorov)?

$$\begin{array}{ll} P(a) = 0{,}4 & P(a \land b) = 0{,}0 \\ P(b) = 0{,}3 & P(a \lor b) = 0{,}8. \end{array} \quad (13.5)$$

Esse tipo de pergunta tem sido objeto de décadas de intenso debate entre aqueles que defendem o uso de probabilidades como a única forma legítima de graus de crença e aqueles que defendem abordagens alternativas.

Um argumento para os axiomas de probabilidade, exposto pela primeira vez em 1931 por Bruno de Finetti [e traduzido para o inglês em Finetti (1993)], é o seguinte: se um agente tem algum grau de crença na proposição a, então o agente deveria ser capaz de exprimir as probabilidades em que é indiferente apostar a favor ou contra a.[3] Pense nisso como um jogo entre dois agentes: o agente 1 afirma: "Meu grau de crença no evento a é de 0,4." O Agente 2 está então livre para escolher se quer apostar a favor ou contra em jogos que são consistentes com o grau de crença declarado. Ou seja, o Agente 2 pode optar por aceitar a aposta do Agente 1 de que a vai ocorrer, oferecendo $6 contra $4 do Agente 1. Ou o Agente 2 pode aceitar a aposta do Agente 1 de que $\neg a$ vai ocorrer, oferecendo $4 contra $6 do Agente 1. Então observamos que o resultado de a e de quem estiver certo recolhe o dinheiro. Se os graus de crença de um agente não refletirem o mundo exatamente, é de se esperar que tendam a perder dinheiro em longo prazo em relação ao agente oponente cujas crenças refletem mais precisamente o estado do mundo.

De Finetti provou algo muito mais forte: *se o Agente 1 expressa um conjunto de graus da crença que violam os axiomas da teoria da probabilidade, há uma combinação de apostas pelo Agente 2, que garante que o Agente 1 vai perder dinheiro toda vez.* Por exemplo, suponha que o Agente 1 tenha o conjunto de graus de crença da Equação 13.5. A Figura 13.2 mostra que, se o Agente 2 escolhe apostar $4 em a, $3 em b e $2 em $\neg(a \lor b)$, então o Agente 1 sempre perderá dinheiro, independentemente dos resultados de a e b. O teorema de de Finetti implica que nenhum agente racional pode ter crenças que violem os axiomas da probabilidade.

Agente 1		Agente 2		Resultados e pagamentos para o Agente 1			
Proposição	Crença	Aposta	Dinheiro apostado	*a, b*	*a,¬b*	*¬a, b*	*¬a,¬b*
a	0,4	*a*	4 a 6	–6	–6	4	4

b	0,3	*b*	3 a 7	–7	3	–7	3
a ∨ *b*	0,8	¬(*a* ∨ *b*)	2 a 8	2	2	2	–8
				–11	–1	–1	–1

Figura 13.2 Devido ao Agente 1 ter crenças inconsistentes, o Agente 2 é capaz de conceber um conjunto de apostas que garante a perda para o Agente 1, não importa o resultado de *a* e *b*.

Uma objeção comum ao teorema de De Finetti é que esse jogo de apostas é um pouco artificial. Por exemplo, o que acontece se alguém se recusa a apostar? Isso acaba com a disputa? A resposta é que o jogo de apostas é um modelo abstrato para a situação de tomada de decisão em que cada agente está *inevitavelmente* envolvido em cada momento. Cada ação (incluindo inatividade) é uma espécie de aposta, e cada resultado pode ser visto como um desenlace da aposta. Recusar-se a apostar é como se recusar a permitir que o tempo passe.

Outros argumentos filosóficos fortes têm sido apresentados pelo uso das probabilidades, principalmente os de Cox (1946), Carnap (1950) e Jaynes (2003). Cada um construiu um conjunto de axiomas para o raciocínio com graus de crenças: não há contradições, correspondência com lógica comum (por exemplo, se a crença em *A* sobe, a crença, em ¬*A* deve baixar), e assim por diante. O único item controverso com relação ao axioma é que os graus de crença devem ser números ou pelo menos agir como números, e devem ser transitivos (se a crença em *A* for maior do que a crença em *B*, que é maior do que a crença em *C*, então a crença em *A* deve ser maior que em *C*) e comparáveis (a crença em *A* deve ser igual, maior ou menor do que a crença em *B*). Então poderá ser provado que a probabilidade é a única abordagem que satisfaz esses axiomas.

No entanto, sendo o mundo do jeito que é, às vezes as demonstrações práticas falam mais alto do que as evidências. O sucesso dos sistemas de raciocínio baseados na teoria da probabilidade tem sido muito eficaz para atrair seguidores. Veremos agora como os axiomas podem ser implantados para fazer inferências.

13.3 INFERÊNCIA COM O USO DE DISTRIBUIÇÕES CONJUNTAS TOTAIS

Nesta seção, descreveremos um método simples de **inferência probabilística**, isto é, a computação de probabilidades posteriores de proposições de consulta dada uma evidência observada. Utilizaremos a distribuição conjunta total como a "base de conhecimento" a partir da qual poderão ser derivadas respostas para todas as perguntas. Ao longo do caminho, também introduziremos várias técnicas úteis para manipular equações que envolvem probabilidades.

	dordedente		¬*dordedente*	
	boticão	¬*boticão*	*boticão*	¬*boticão*
cárie	0,108	0,012	0,072	0,008
¬*cárie*	0,016	0,064	0,144	0,576

Figura 13.3 Uma distribuição conjunta total para o mundo de *DorDeDente*, *Cárie*, *Boticão*.

Começaremos com um exemplo muito simples: um domínio que consiste apenas nas três variáveis booleanas *DorDeDente*, *Cárie* e *Boticão* (a horrível tenaz de aço com que o dentista agarra meu dente para extraí-lo). A distribuição conjunta total é uma tabela 2 × 2 × 2, como mostra a Figura 13.3.

Note que as probabilidades na distribuição conjunta têm a soma 1, conforme exigem os axiomas de probabilidade. Note também que a Equação 13.2 fornece um caminho direto para calcular a probabilidade de qualquer proposição, simples ou complexa: simplesmente identificamos os mundos possíveis nos quais a proposição é verdadeira e somamos suas probabilidades. Por exemplo, existem seis eventos atômicos em que *cárie* ∨ *dordedente* é válida:

$$P(cárie \lor dordedente) = 0{,}108 + 0{,}012 + 0{,}072 + 0{,}008 + 0{,}016 + 0{,}064 = 0{,}28.$$

Uma tarefa particularmente comum é extrair a distribuição sobre algum subconjunto de variáveis ou sobre uma única variável. Por exemplo, a adição das entradas da primeira linha produz a probabilidade incondicional ou **probabilidade marginal**[4] de *cárie*:

$$P(cárie) = 0{,}108 + 0{,}012 + 0{,}072 + 0{,}008 = 0{,}2.$$

DE ONDE VÊM AS PROBABILIDADES?

Existe um debate sem fim sobre a origem e o *status* de valores de probabilidade. A posição **frequentista** afirma que os números só podem vir de *experimentos*: se testarmos 100 pessoas e descobrirmos que 10 delas têm cáries, poderemos afirmar que a probabilidade de uma cárie é aproximadamente 0,1. Segundo essa visão, a asserção “a probabilidade de uma cárie é 0,1” significa que 0,1 é a fração que seria observada no limite de infinitamente muitas amostras. A partir de qualquer amostra finita, podemos estimar a fração verdadeira e também calcular a probabilidade de exatidão de nossa estimativa.

A visão **objetivista** afirma que as probabilidades são aspectos reais do universo — tendências de objetos a se comportarem de determinadas maneiras —, em vez de serem apenas descrições do grau de crença de um observador. Por exemplo, o fato de o lançamento de uma moeda comum dar cara com probabilidade 0,5 é uma tendência da própria moeda. Segundo essa visão, as medições frequentistas são tentativas de observar essas tendências. A maior parte dos físicos concorda com o fato de que os fenômenos quânticos são objetivamente probabilísticos, mas a incerteza na escala macroscópica — por exemplo, no lançamento de uma moeda — em geral surge da ignorância das condições iniciais e não parece consistente com a visão de propensão.

A visão **subjetivista** descreve probabilidades como um modo de caracterizar as crenças de um agente, em vez de ter qualquer significado físico externo. A visão subjetiva **bayesiana** permite qualquer atribuição autoconsistente de probabilidades anteriores a proposições, mas insiste na própria atualização bayesiana à medida que a evidência chega.

No final, até mesmo uma posição frequentista rígida envolve análise subjetiva, por causa da

classe de referência: na tentativa de determinar a probabilidade de resultado de uma experiência *particular*, o frequentista tem de colocá-la em uma classe de referência de experimentos "similares" com frequências de resultados conhecidos. I. J. Good (1983, p. 27) escreveu: "Todos os eventos na vida são únicos, e toda probabilidade da vida real que estimamos na prática é a de um evento que nunca ocorreu antes." Por exemplo, dado um paciente em particular, um frequentista que deseja estimar a probabilidade de uma cárie, vai considerar uma classe de referência de outros pacientes que são semelhantes em itens importantes — idade, sintomas, dieta — e verificar que proporção deles tem uma cárie. Se o médico considerar tudo o que se conhece sobre o paciente — o peso aproximado até o grama mais próximo, cor do cabelo, nome de solteira da mãe etc. —, o resultado será a inexistência de outros pacientes exatamente iguais e, portanto, não haverá nenhuma classe de referência a partir da qual coletar dados experimentais. Esse foi um problema constrangedor na filosofia da ciência.

O **princípio da indiferença** atribuído a Laplace (1816) declara que as proposições sintaticamente "simétricas" com relação à evidência devem ser consideradas proposições de igual probabilidade. Vários aprimoramentos foram propostos, culminando na tentativa de Carnap e outros de desenvolver uma rigorosa **lógica indutiva**, capaz de calcular a probabilidade correta para qualquer proposição a partir de qualquer coleção de observações. Atualmente, acredita-se que não existe nenhuma lógica indutiva única; em vez disso, qualquer lógica desse tipo se baseia em uma distribuição subjetiva de probabilidade *a priori* cujo efeito é reduzido à medida que são coletados resultados de outras observações.

Esse processo é chamado **marginalização** ou **totalização** porque totalizamos as probabilidades para cada valor possível de outras variáveis, assim excluindo-as da equação. Podemos escrever a regra geral de marginalização a seguir para quaisquer conjuntos de variáveis **Y** e **Z**:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z} \in \mathbf{Z}} \mathbf{P}(\mathbf{Y}, \mathbf{z}) \,, \tag{13.6}$$

onde $\Sigma_{\mathbf{z} \in \mathbf{Z}}$ significa a soma sobre todas as combinações possíveis de valores do conjunto de variáveis **Z**. Por vezes, abreviamos como $\Sigma_{\mathbf{z}}$, deixando **Z** implícito. Apenas utilizamos a regra como

$$\mathbf{P}(\mathit{Cárie}) = \sum_{\mathbf{z} \in \{\mathit{Boticão},\, \mathit{Dor\ de\ dente}\}} \mathbf{P}(\mathit{Cárie}, \mathbf{z}) \,. \tag{13.7}$$

Uma variante dessa regra envolve probabilidades condicionais em vez de probabilidades conjuntas, usando-se a regra do produto:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y} \mid \mathbf{z}) P(\mathbf{z}) \,. \tag{13.8}$$

Essa regra é chamada **condicionamento**. Marginalização e condicionamento se mostrarão regras úteis para todos os tipos de derivações que envolverem expressões de probabilidade.

Na maioria dos casos, estaremos interessados em calcular probabilidades *condicionais* de

algumas variáveis, dada alguma evidência sobre outras. As probabilidades condicionais podem ser descobertas usando-se primeiro a Equação 13.3 para obter uma expressão em termos de probabilidades não condicionais, e então avaliando-se a expressão a partir da distribuição conjunta total. Por exemplo, podemos calcular a probabilidade de uma cárie, dada a evidência de uma dor de dente, como a seguir:

$$P(cárie \mid dordedente) = \frac{P(cárie \wedge dordedente)}{P(dordedente)}$$
$$= \frac{0{,}108 + 0{,}012}{0{,}108 + 0{,}012 + 0{,}016 + 0{,}064} = 0{,}6.$$

Só para conferir, também podemos calcular a probabilidade de não haver nenhuma cárie, dada uma dor de dente:

$$P(\neg cárie \mid dordedente) = \frac{P(\neg cárie \wedge dordedente)}{P(dordedente)}$$
$$= \frac{0{,}016 + 0{,}064}{0{,}108 + 0{,}012 + 0{,}016 + 0{,}064} = 0{,}4.$$

Conforme esperado, as duas variáveis somam 1. Note que, nesses dois cálculos, a expressão $1/P(dordedente)$ permanece constante, não importando que valor de *Cárie* calculamos. De fato, ela pode ser visualizada como uma constante de **normalização** para a distribuição $\mathbf{P}(Cárie \mid dordedente)$, assegurando que a soma será 1. Ao longo dos capítulos que lidam com probabilidade, usaremos α *para* denotar tais constantes. Com essa notação, podemos transformar as duas equações precedentes em uma:

$$\mathbf{P}(Cárie \mid dordedente) = \alpha \mathbf{P}(Cárie, dordedente)$$
$$= \alpha\, [\mathbf{P}(Cárie, dordedente, boticão) + \mathbf{P}(Cárie, dordedente, \neg boticão)]$$
$$= \alpha\, [\langle 0{,}108, 0{,}016\rangle + \langle 0{,}012, 0{,}064\rangle] = \alpha\, \langle 0{,}12, 0{,}08\rangle = \langle 0{,}6, 0{,}4\rangle.$$

Em outras palavras, podemos calcular $\mathbf{P}(Cárie \mid dor\ de\ dente)$ mesmo se não soubermos o valor de $\mathbf{P}(dor\ de\ dente)$! Esquecemos temporariamente o fator $1/P(dor\ de\ dente)$ e somamos os valores de cárie e ¬*cárie*, obtendo 0,12 e 0,08. Essas são as proporções relativas corretas, mas não perfazem 1, de modo que as normalizamos, dividindo cada uma por 0,12 + 0,08, ficando com as probabilidades verdadeiras de 0,6 e 0,4. A normalização acaba por ser um atalho útil em muitos cálculos de probabilidade, tanto para tornar a computação mais fácil como para permitir-nos continuar quando alguma avaliação de probabilidade (como $\mathbf{P}(dor\ de\ dente)$) não estiver disponível.

A partir do exemplo, podemos extrair um procedimento de inferência geral. Começamos com o caso em que a consulta envolve uma única variável, X (*Cárie* no exemplo). Seja $\mathbf{E}$ o conjunto de variáveis de evidência (apenas *DorDeDente* no exemplo), seja $\mathbf{e}$ o conjunto de valores observados para elas e seja $\mathbf{Y}$ as variáveis restantes não observadas (apenas *Boticão* no exemplo). A consulta é $\mathbf{P}(X \mid \mathbf{e})$ e pode ser avaliada como:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha\, \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}), \tag{13.9}$$

onde o somatório é efetuado sobre todos os valores **y** possíveis (isto é, todas as combinações possíveis de valores das variáveis não observadas **Y**). Note que, juntas, as variáveis *X*, **E** e **Y** constituem o conjunto completo de variáveis para o domínio; assim, **P**(*X*, **e**, **y**) é simplesmente um subconjunto de probabilidades a partir da distribuição conjunta total.

Dada a distribuição conjunta total de trabalho, a Equação 13.9 pode responder a consultas probabilísticas referentes a variáveis discretas. Porém, ele não aumenta de escala muito bem: para um domínio descrito por *n* variáveis booleanas, o algoritmo exige uma tabela de entrada com o tamanho $O(2^n)$ e demora o tempo $O(2^n)$ para processar a tabela. Em um problema realista, podemos ter facilmente $n > 100$, tornando $O(2^n)$ impraticável. Por essas razões, a distribuição conjunta total em forma tabular não é uma ferramenta prática para construir sistemas de raciocínio.

Em vez disso, devemos visualizá-la como o fundamento teórico sobre o qual podem ser elaboradas abordagens mais efetivas, assim como as tabelas de verdade formaram um fundamento teórico para os algoritmos mais práticos como DPLL. O restante deste capítulo introduz algumas das ideias básicas necessárias na preparação para o desenvolvimento de sistemas realistas, no Capítulo 14.

13.4 INDEPENDÊNCIA

Vamos expandir a distribuição conjunta total da Figura 13.3, adicionando uma quarta variável, *Tempo*. A distribuiçao conjunta total então se torna **P**(*DorDeDente*, *Boticão*, *Cárie*, *Tempo*), que tem 2 × 2 × 2 × 4 = 32 entradas. Ela contém quatro "edições" da tabela mostrada na Figura 13.3, uma para cada espécie de tempo. Parece natural indagar que relacionamento essas edições mantêm umas com as outras e com a tabela original de três variáveis. Por exemplo, como *P*(*dordedente*, *boticão*, *cárie*, *nublado*) e *P*(*dordedente*, *boticão*, *cárie*) estão relacionadas? Podemos utilizar a regra do produto:

$$P(dordedente, boticão, cárie, nublado)$$
$$= P(nublado \mid dordedente, boticão, cárie)\, P(dordedente, boticão, cárie).$$

Agora, a menos que se esteja no ramo comercial de divindade, não se deve imaginar que os problemas dentários de alguém influenciam as condições do tempo. E, para a odontologia em consultório, pelo menos, parece seguro dizer que o tempo não influencia as variáveis dentárias. Portanto, a asserção a seguir parece razoável:

$$P(nublado \mid dordedente, boticão, cárie) = P(nublado). \qquad (13.10)$$

A partir disso, podemos deduzir:

$$P(dordedente, boticão, cárie, nublado) = P(nublado)\, P(dordedente, boticão, cárie).$$

Existe uma equação semelhante para *toda entrada* em **P**(*DorDeDente, Boticão*, *Cárie*, *Tempo*). De fato, podemos escrever a equação geral:

$$\mathbf{P}(DorDeDente, Boticão, Cárie, Tempo) = \mathbf{P}(DorDeDente, Boticão, Cárie)\mathbf{P}(Tempo).$$

Desse modo, a tabela de 32 elementos para quatro variáveis pode ser construída a partir de uma tabela de oito elementos e uma tabela de quatro elementos. Essa decomposição é ilustrada esquematicamente na Figura 13.4(a).

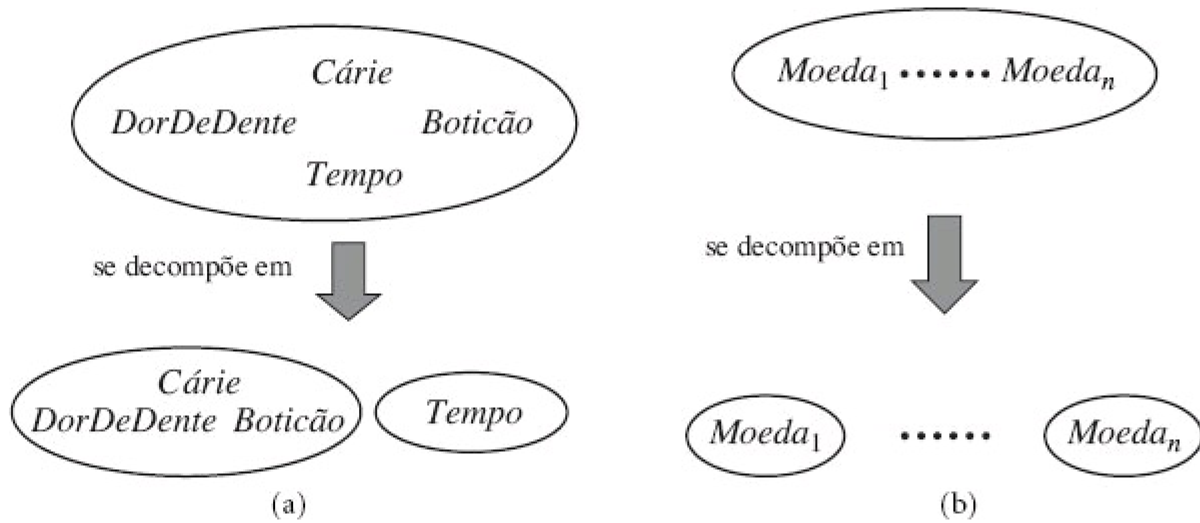


Figura 13.4 Dois exemplos de fatoração de uma grande distribuição conjunta em distribuições menores, com a utilização da independência absoluta. (a) As condições do tempo e os problemas dentários são independentes. (b) Lançamentos de moedas são independentes.

A propriedade que empregamos para escrever a Equação 13.10 é chamada **independência** (também **independência marginal** e **independência absoluta**). Em particular, o tempo é independente dos problemas dentários de alguém. A independência entre as proposições *a* e *b* pode ser escrita como:

$$P(a \mid b) = P(a) \text{ ou } P(b \mid a) = P(b) \text{ ou } P(a \wedge b) = P(a)P(b). \quad (13.11)$$

Todas essas formas são equivalentes (Exercício 13.12). A independência entre as variáveis X e Y pode ser escrita como a seguir (mais uma vez, essas formas são todas equivalentes):

$$\mathbf{P}(X \mid Y) = \mathbf{P}(X) \text{ ou } \mathbf{P}(Y \mid X) = \mathbf{P}(Y) \text{ ou } \mathbf{P}(X, Y) = \mathbf{P}(X)\mathbf{P}(Y).$$

As asserções de independência em geral se baseiam no conhecimento do domínio. Como o exemplo do tempo na dor de dente ilustra, elas podem reduzir drasticamente a quantidade de informações necessárias para especificar a distribuição conjunta total. Se o conjunto completo de variáveis puder ser dividido em subconjuntos independentes, então a distribuição conjunta total poderá ser *fatorada* em distribuições conjuntas separadas sobre esses subconjuntos. Por exemplo, a distribuição conjunta total sobre o resultado de n lançamentos de moedas independentes, $\mathbf{P}(C1, \ldots, Cn)$, tem 2^n entradas, mas pode ser representada como o produto de n distribuições de variáveis $\mathbf{P}(C_i)$. De modo mais prático, a independência entre a odontologia e a meteorologia é algo bom porque, do contrário, a prática da odontologia poderia exigir o conhecimento íntimo da meteorologia e *vice-versa*.

Portanto, quando estão disponíveis, as asserções de independência podem ajudar na redução do

tamanho da representação do domínio e da complexidade do problema de inferência. Infelizmente, a separação clara e completa de conjuntos de variáveis por independência é bastante rara. Sempre que existir uma conexão, ainda que indireta entre duas variáveis, a independência deixará de ser válida. Além disso, até mesmo subconjuntos independentes podem ser bastante grandes — por exemplo, a odontologia pode envolver dezenas de doenças e centenas de sintomas, todos inter-relacionados. Para tratar de tais problemas, precisaremos de métodos mais sutis que o simples conceito de independência.

13.5 A REGRA DE BAYES E SEU USO

Anteriormente, definimos a **regra do produto.** Ela pode ser escrita de duas formas:

$$P(a \wedge b) = P(a \mid b)P(b) \qquad \text{e} \qquad P(a \wedge b) = P(b \mid a)P(a).$$

Igualando os dois membros da direita e dividindo por $P(a)$, obtemos:

$$P(b \mid a) = \frac{P(a \mid b)P(b)}{P(a)}. \tag{13.12}$$

Essa equação é conhecida como **regra de Bayes** (e também como lei de Bayes ou teorema de Bayes). Essa equação simples é a base de todos os sistemas modernos de IA para inferência probabilística.

O caso mais geral de variáveis multivaloradas pode ser escrito na notação de **P** como segue:

$$\mathbf{P}(Y \mid X) = \frac{\mathbf{P}(X \mid Y)\mathbf{P}(Y)}{\mathbf{P}(X)},$$

Como antes, essa equação deve ser considerada a representação de um conjunto de equações, cada uma lidando com valores específicos das variáveis. Também teremos a oportunidade de usar uma versão mais geral condicionalizada em alguma evidência prática **e**:

$$\mathbf{P}(Y \mid X, \mathbf{e}) = \frac{\mathbf{P}(X \mid Y, \mathbf{e})\mathbf{P}(Y \mid \mathbf{e})}{\mathbf{P}(X \mid \mathbf{e})}. \tag{13.13}$$

13.5.1 Aplicação da regra de Bayes: o caso simples

À primeira vista, a regra de Bayes não parece muito útil. Ela nos permite calcular o único termo $P(b \mid a)$ em termos de três termos: $P(a \mid b)$, $P(b)$ e $P(a)$. Isso parece como a duas etapas atrás, mas a regra de Bayes é útil na prática porque existem muitos casos em que fazemos boas estimativas de probabilidade para esses três números e precisamos calcular o quarto número. Muitas vezes, percebemos o *efeito* como evidência de alguma *causa* desconhecida e gostaríamos de determinar essa causa. Nesse caso, a regra de Bayes torna-se

$$P(causa \mid efeito) = \frac{P(efeito \mid causa)\, P(causa)}{P(efeito)}.$$

A probabilidade condicional $P(efeito|causa)$ quantifica a relação na direção **causal**, enquanto $P(causa|efeito)$ descreve a direção do **diagnóstico**. Em uma tarefa como o diagnóstico médico com frequência temos probabilidades condicionais sobre relacionamentos causais (isto é, o médico conhece $P(sintomas|doenças)$) e quer derivar um diagnóstico $P(doenças|sintomas)$. Por exemplo, um médico sabe que a meningite faz o paciente ter uma rigidez no pescoço, digamos, durante 70% do tempo. O médico também conhece alguns fatos incondicionais: a probabilidade *a priori* de um paciente ter meningite é 1/50.000, e a probabilidade *a priori* de qualquer paciente ter rigidez no pescoço é 1%. Sendo *s* a proposição de que o paciente tem rigidez no pescoço e *m* a proposição de que o paciente tem meningite, temos:

$$\begin{aligned} P(s \mid m) &= 0{,}7 \\ P(m) &= 1/50000 \\ P(s) &= 0{,}01 \\ P(m \mid s) &= \frac{P(s \mid m)P(m)}{P(s)} = \frac{0{,}7 \times 1/50000}{0{,}01} = 0{,}0014\,. \end{aligned} \tag{13,14}$$

Ou seja, esperamos que apenas um em 5.000 pacientes com rigidez no pescoço tenha meningite. Note que, embora a rigidez no pescoço seja uma indicação bastante forte de meningite (com probabilidade 0,7), a probabilidade de o paciente estar acometido de meningite permanece pequena. Isso ocorre porque a probabilidade *a priori* sobre rigidez no pescoço é muito mais alta que a probabilidade *a priori* sobre meningite.

A Seção 13.3 ilustrou um processo pelo qual é possível evitar a avaliação da probabilidade da evidência (aqui, $P(s)$), calculando-se em vez disso uma probabilidade posterior para cada valor da variável de consulta (nesse caso, m e $\neg m$) e depois normalizando-se os resultados. O mesmo processo pode ser aplicado quando se utiliza a regra de Bayes. Temos:

$$\mathbf{P}(M \mid s) = a\, \langle P(s \mid m)P(m), P(s \mid \neg m)P(\neg m)\rangle.$$

Desse modo, para utilizar essa abordagem, precisamos fazer uma estimativa de $P(s \mid \neg m)$ em vez de $P(s)$. Não existe almoço grátis — às vezes é mais fácil, às vezes mais difícil. A forma geral da regra de Bayes com normalização é:

$$\mathbf{P}(Y|X) = \alpha\, \mathbf{P}(X|Y)\mathbf{P}(Y), \tag{13.15}$$

onde α é a constante de normalização necessária para fazer as entradas de $\mathbf{P}(Y \mid X)$ terem soma igual a 1.

Uma pergunta óbvia sobre a regra de Bayes é por que a probabilidade condicional pode estar disponível em um sentido, mas não no outro. No domínio de meningite, talvez o médico saiba que a rigidez no pescoço implica meningite em um entre 5.000 casos; isto é, o médico tem informações quantitativas no sentido do **diagnóstico** de sintomas para causas. Tal médico não tem necessidade de usar a regra de Bayes. Infelizmente, *o conhecimento do diagnóstico frequentemente é mais frágil que o conhecimento causal*. Se houver uma súbita epidemia de meningite, a probabilidade

incondicional de meningite, $P(m)$, crescerá. O médico que derivou a probabilidade de diagnóstico $P(m \mid s)$ diretamente da observação estatística de pacientes antes da epidemia não terá ideia de como atualizar o valor, mas o médico que calcular $P(m \mid s)$ a partir dos outros três valores verá que $P(m \mid s)$ deve subir proporcionalmente com $P(m)$. Mais importante ainda, as informações causais $P(s \mid m)$ *não são afetadas* pela epidemia porque simplesmente refletem o modo como a meningite atua. O uso dessa espécie de conhecimento causal ou baseado em modelos fornece a robustez crucial necessária para tornar os sistemas probabilísticos viáveis no mundo real.

13.5.2 Utilização da regra de Bayes: combinação de evidências

Vimos que a regra de Bayes pode ser útil para responder a consultas probabilísticas condicionadas sobre uma única peça de evidência — por exemplo, a rigidez no pescoço. Em particular, argumentamos que as informações probabilísticas com frequência estão disponíveis sob a forma $P(efeito|causa)$. O que acontece quando temos duas ou mais peças de evidência? Por exemplo, o que um dentista pode concluir se seu terrível boticão agarrar o dente dolorido de um paciente? Se conhecermos a distribuição conjunta total (Figura 13.3), poderemos representar a resposta:

$$\mathbf{P}(Cárie|dordedente \wedge boticão) = \alpha \langle 0{,}108,\ 0{,}016 \rangle \approx \langle 0{,}871,\ 0{,}129 \rangle.$$

Porém, sabemos que tal abordagem não poderá aumentar de escala até quantidades maiores de variáveis. Podemos tentar utilizar a regra de Bayes para reformular o problema:

$$\mathbf{P}(Cárie|dordedente \wedge boticão) = \alpha \mathbf{P}(dordedente \wedge boticão|Cárie)\ \mathbf{P}(Cárie). \qquad (13.16)$$

Para essa reformulação funcionar, precisamos conhecer as probabilidades condicionais da conjunção *dordedente* ∧ *boticão* para cada valor de *Cárie*. Isso poderia ser viável para apenas duas variáveis de evidência, mas de novo não aumentará de escala. Se houvessem n variáveis de evidência possíveis (raios X, dieta, higiene oral etc.), então haveria 2^n combinações possíveis de valores observados para os quais precisaríamos conhecer probabilidades condicionais. Também poderíamos voltar a usar a distribuição conjunta total. Isso foi o que primeiro afastou os pesquisadores da teoria da probabilidade, levando-os a estudar métodos aproximados para combinação de evidências que, embora forneçam respostas incorretas, exigem menor quantidade de números para fornecer alguma resposta.

Em vez de seguir esse caminho, precisamos encontrar algumas asserções adicionais sobre o domínio que nos permitirão simplificar as expressões. A noção de **independência** da Seção 13.4 fornece uma pista, mas precisa de refinamento. Seria bom se *DorDeDente* e *Boticão* fossem independentes, mas não são: se a ferramenta agarrar o dente, isso significa que o dente tem uma cárie que deve provocar uma dor de dente. No entanto, essas variáveis *são* independentes, *dada a presença ou a ausência de uma cárie*. Cada uma é diretamente causada pela cárie, mas nenhuma delas tem efeito direto sobre a outra: a dor de dente depende do estado dos nervos do dente, enquanto a precisão da ferramenta depende da habilidade do dentista, para o qual a dor de dente é irrelevante.[5] Matematicamente, essa propriedade é escrita como:

$$\mathbf{P}(dordedente \wedge boticão | Cárie) = \mathbf{P}(dordedente | Cárie)\mathbf{P}(boticão | Cárie). \quad (13.17)$$

Essa equação expressa a **independência condicional** de *dordedente* e *boticão* dada *Cárie*. Podemos inseri-la na Equação 13.16 para obter a probabilidade de uma cárie:

$$\mathbf{P}(Cárie | dordedente \wedge boticão) = \alpha P(dordedente | Cárie)\, \mathbf{P}(boticão | Cárie)\, \mathbf{P}(Cárie). \quad (13.18)$$

Agora, os requisitos de informações são idênticos aos da inferência, utilizando cada item de evidência separadamente: a probabilidade *a priori* **P**(*Cárie*) para a variável de consulta e a probabilidade condicional de cada efeito, dada sua causa.

A definição geral de **independência condicional** de duas variáveis *X* e *Y*, dada uma terceira variável *Z,* é:

$$\mathbf{P}(X, Y \mid Z) = \mathbf{P}(X \mid Z)\mathbf{P}(Y \mid Z).$$

Por exemplo, no domínio de dentista, é razoável assumir a independência condicional das variáveis *DorDeDente* e *boticão*, dada *Cárie*:

$$\mathbf{P}(DorDeDente, Boticão | Cárie) = \mathbf{P}(DorDeDente | Cárie)\mathbf{P}(Boticão | Cárie). \quad (13.19)$$

Note que essa asserção é um pouco mais forte que a Equação 13.17, que afirma a independência apenas para valores específicos de *DorDeDente* e *Boticão*. Como ocorre com a independência absoluta na Equação 13.11, as formas equivalentes

$$\mathbf{P}(X \mid Y, Z) = \mathbf{P}(X \mid Z) \text{ e } \mathbf{P}(Y \mid X, Z) = \mathbf{P}(Y \mid Z)$$

também podem ser usadas (veja o Exercício 13.17). A Seção 13.17 mostrou que as asserções de independência absoluta permitem uma decomposição da distribuição conjunta total em itens muito menores. Ocorre que o mesmo é verdadeiro para asserções de independência condicional. Por exemplo, dada a Equação 13.19, podemos derivar uma decomposição como:

$$\begin{aligned}&\mathbf{P}(DorDeDente, Boticão, Cárie)\\ &\quad = \mathbf{P}(DorDeDente, Boticão \mid Cárie)\mathbf{P}(Cárie) \text{ (regra do produto)}\\ &\quad = \mathbf{P}(DorDeDente \mid Cárie)\mathbf{P}(Boticão \mid Cárie)\mathbf{P}(Cárie) \text{ (usando 13.19).}\end{aligned}$$

☞ (Na Figura 13.3, o leitor pode verificar que realmente essa equação é válida.) Desse modo, a grande tabela original é decomposta em três tabelas menores. A tabela original tem sete números independentes ($2^3 = 8$ entradas na tabela, mas elas devem somar 1, então 7 são independentes). As tabelas menores contêm cinco números independentes (para distribuições de probabilidade condicional como o **P**($T \mid C$), há duas linhas de dois números, e cada linha soma 1, de modo que são dois números independentes; para uma distribuição *a priori*, como **P**(C), há somente um número independente.) Ir de 7 até 5 pode não parecer um grande triunfo, mas a verdade é que, para n sintomas que são condicionalmente independentes dada *Cárie*, o tamanho da representação cresce como $O(n)$ e não como $O(2^n)$. Isso significa que *as asserções de independência condicional podem*

permitir o aumento da escala de sistemas probabilísticos; *além disso*, *elas são muito mais comuns que as asserções de independência absoluta*. Conceitualmente, *Cárie* **separa** *DorDeDente* e *Boticão* porque é uma causa direta de ambas. A decomposição de grandes domínios probabilísticos em subconjuntos conectados livremente por meio de independência condicional é um dos desenvolvimentos mais importantes na história recente da IA.

O exemplo de odontologia ilustra um padrão que ocorre comumente, no qual uma única causa influencia de maneira direta vários efeitos, todos condicionalmente independentes, dada a causa. A distribuição conjunta total pode ser escrita como:

$$\mathbf{P}(Causa, Efeito_1, \ldots, Efeito_n) = \mathbf{P}(Causa) \prod_i \mathbf{P}(Efeito_n \mid Causa) .$$

Tal distribuição de probabilidade é chamada modelo bayesiano **ingênuo** — "ingênuo" porque é usado com frequência (como uma hipótese simplificadora) em casos nos quais as variáveis "efeito" *não* são na realidade condicionalmente independentes dada a variável causa. (Algumas vezes, o modelo de Bayes ingênuo é chamado **classificador de Bayes**, um uso um tanto descuidado que levou os verdadeiros seguidores de Bayes a denominá-lo modelo de **Bayes idiota**.) Na prática, sistemas de Bayes ingênuos podem funcionar surpreendentemente bem, mesmo quando a hipótese de independência não é verdadeira. O Capítulo 20 descreve métodos para aprendizado de distribuições de Bayes ingênuas a partir de observações.

13.6 DE VOLTA AO MUNDO DE WUMPUS

Podemos combinar muitas das ideias deste capítulo para resolver problemas de raciocínio probabilístico no mundo de wumpus (veja no Capítulo 7 uma descrição completa do mundo de wumpus). A incerteza surge no mundo de wumpus porque os sensores do agente fornecem apenas informações parciais locais sobre o mundo. Por exemplo, a Figura 13.5 mostra uma situação em que cada um dos três quadrados acessíveis — [1,3], [2,2] e [3,1] — poderia conter um poço. A inferência lógica pura não pode concluir nada sobre qual quadrado tem maior probabilidade de ser seguro e, assim, um agente lógico poderia ser forçado a escolher ao acaso.

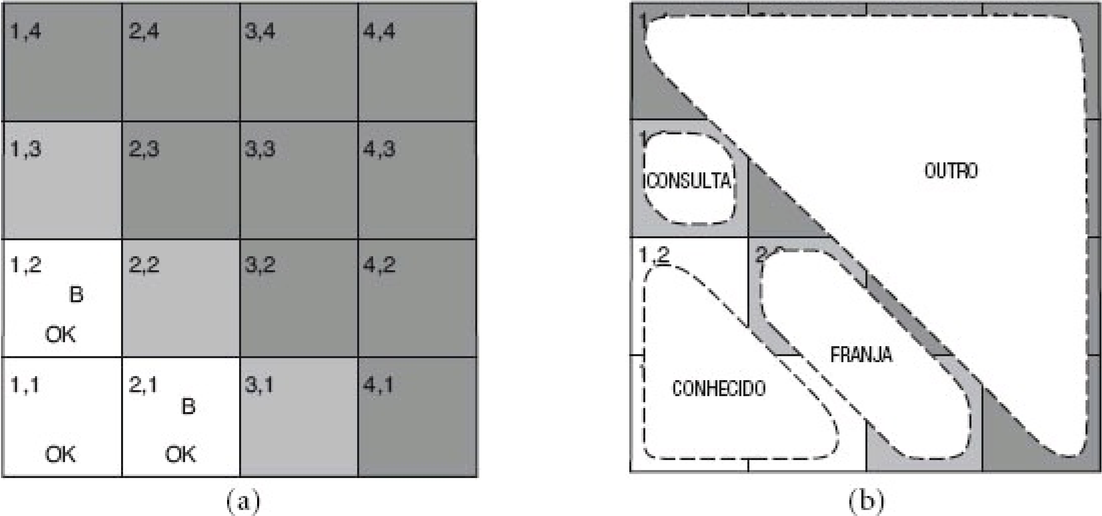


Figura 13.5 (a) Depois de encontrar uma brisa em[1,2] e [2,1], o agente está paralisado — não existe

lugar seguro para explorar. (b) Divisão dos quadrados em *Conhecido*, *Franja* e *Outro*, para uma consulta sobre [1,3].

Veremos que um agente probabilístico pode fazer muito melhor que o agente lógico.

Nosso objetivo será calcular a probabilidade de que cada um dos três quadrados contenha um poço (para os propósitos deste exemplo, ignoraremos o wumpus e o ouro). As propriedades relevantes do mundo de wumpus são: (1) um poço causa brisas em todos os quadrados vizinhos e (2) cada quadrado diferente de [1,1] contém um poço com probabilidade 0,2. O primeiro passo é identificar o conjunto de variáveis aleatórias de que necessitamos:

- Como no caso da lógica proposicional, queremos usar uma variável booleana P_{ij} para cada quadrado, o que é verdadeiro se e somente se o quadrado $[i, j]$ realmente contém um poço.
- Também temos variáveis booleanas B_{ij} que são verdadeiras se e somente se o quadrado $[i, j]$ for arejado; incluímos essas variáveis apenas para os quadrados observados — nesse caso, [1,1], [1,2] e [2,1].

O próximo passo é especificar a distribuição conjunta total, $\mathbf{P}(P_{1,1}, \ldots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1})$. Aplicando a regra do produto, temos:

$$\mathbf{P}(P_{1,1},\ldots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1}) = \\ \mathbf{P}(B_{1,1}, B_{1,2}, B_{2,1} | P_{1,1},\ldots, P_{4,4})\, \mathbf{P}(P_{1,1},\ldots, P_{4,4}).$$

Essa decomposição torna muito fácil ver quais devem ser os valores de probabilidade conjunta. O primeiro termo é a probabilidade condicional de uma configuração de brisa, dada uma configuração de poço; seus valores são 1 se as brisas são adjacentes aos poços e 0 em caso contrário. O segundo termo é a probabilidade *a priori* de uma configuração de poço. Cada quadrado contém um poço com probabilidade 0,2, independentemente dos outros quadrados; por conseguinte,

$$\mathbf{P}(P_{1,1}, \ldots, P_{4,4}) = \prod_{i,j=1,1}^{4,4} \mathbf{P}(P_{i,j}). \tag{13.20}$$

Para uma configuração com n poços, $P(P_{1,1},\ldots,P_{4,4}) = 0{,}2^{n}\ 0{,}8^{16-n}$.

Na situação da Figura 13.5(a), a evidência consiste na brisa observada (ou em sua ausência) em cada quadrado visitado, combinada com o fato de que cada quadrado não contém nenhum poço. Abreviaremos esses fatos como $b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$ e $conhecido = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$. Estamos interessados em responder a consultas tais como $\mathbf{P}(P_{1,3} | conhecido, b)$: qual é a probabilidade de que [1,3] contenha um poço, dadas as observações até agora?

Para responder a essa consulta, podemos seguir a abordagem-padrão da Equação 13.9, ou seja, efetuar o somatório sobre todas as entradas da distribuição conjunta total. Seja *Desconhecido* uma variável composta que consiste nas $P_{i,j}$ variáveis correspondentes a outros quadrados que não os quadrados assinalados com *Conhecido* e no quadrado de consulta [1,3]. Então, pela Equação 13.9,

temos:

$$\mathbf{P}(P_{1,3} | conhecido, b) = \alpha \sum_{desconhecido} \mathbf{P}(P_{1,3}, desconhecido, conhecido, b).$$

As probabilidades conjuntas totais já foram especificadas e, portanto, terminamos — isto é, a não ser que estejamos preocupados com a computação. Existem 12 quadrados desconhecidos; consequentemente, o somatório contém 2^{12} = 4.096 termos. Em geral, o somatório cresce exponencialmente com o número de quadrados.

Poderíamos perguntar se os outros quadrados certamente não são irrelevantes. Como [4,4] afetaria o fato de [1,3] ter um poço? Na realidade, essa intuição está correta. Façamos a *Fronteira* ser as variáveis (diferentes da variável de consulta) que são adjacentes a quadrados visitados, nesse caso apenas [2,2] e [3,1]. Além disso, sejam *Outro* as variáveis para os outros quadrados desconhecidos; nesse caso, existem 10 outros quadrados, como mostra a Figura 13.5(b). A ideia-chave é que as brisas observadas são *condicionalmente independentes* das outras variáveis, dados conhecido, fronteira e variáveis de consulta. Para usar essa ideia, manipulamos a fórmula de consulta em uma forma na qual as brisas são condicionadas sobre todas as outras variáveis e depois aplicamos a independência condicional:

$$\begin{aligned}
&\mathbf{P}(P_{1,3} | conhecido, b) \\
&= \alpha \sum_{desconhecido} \mathbf{P}(P_{1,3} | conhecido, b, desconhecido) \qquad \text{(pela Equação (13.9))} \\
&= \alpha \sum_{desconhecido} \mathbf{P}(b | P_{1,3}, conhecido, desconhecido)\mathbf{P}(P_{1,3}, conhecido, desconhecido) \\
&\qquad \text{(peça regra do produto)} \\
&= \alpha \sum_{fronteira} \sum_{outro} \mathbf{P}(b | conhecido, P_{1,3}, fronteira, outro)\mathbf{P}(P_{1,3}, conhecido, fronteira, outro) \\
&= \alpha \sum_{fronteira} \sum_{outro} \mathbf{P}(b | conhecido, P_{1,3}, fronteira)\mathbf{P}(P_{1,3}, conhecido, fronteira, outro),
\end{aligned}$$

onde a última etapa utiliza independência condicional: *b* é independente de *outro* dado *conhecido*, $P_{1,3}$, e *fronteira*. Agora, o primeiro termo nessa expressão não depende das outras variáveis; então podemos mover o somatório para o interior:

$$\begin{aligned}
&\mathbf{P}(P_{1,3} | conhecido, b) \\
&= \alpha \sum_{fronteira} \mathbf{P}(b | conhecido, P_{1,3}, fronteira) \sum_{outro} \mathbf{P}(P_{1,3}, conhecido, fronteira, outro).
\end{aligned}$$

Por independência, como na Equação 13.20, a expressão anterior pode ser fatorada e, portanto, é possível reordenar os termos:

$$\mathbf{P}(P_{1,3} \mid conhecido, b)$$

$$= \alpha \sum_{fronteira} \mathbf{P}(b \mid conhecido, P_{1,3}, fronteira) \sum_{outro} \mathbf{P}(P_{1,3})\, P(conhecido)\, P(fronteira)\, P(outro)$$

$$= \alpha\, P(conhecido)\mathbf{P}(P_{1,3}) \sum_{fronteira} \mathbf{P}(b \mid conhecido, P_{1,3}, fronteira)\, P(fronteira) \sum_{outro} P(outro)$$

$$= \alpha'\, \mathbf{P}(P_{1,3}) \sum_{fronteira} \mathbf{P}(b \mid conhecido, P_{1,3}, fronteira)\, P(fronteira),$$

onde o último passo incorpora $P(conhecido)$ à constante de normalização e utiliza o fato de que $\sum_{outro} P(outro)$ é igual a 1.

Agora, só existem quatro termos no somatório sobre as variáveis de franja $P_{2,2}$ e $P_{3,1}$. O uso da independência e da independência condicional eliminou completamente de consideração os outros quadrados.

Observe que a expressão $\mathbf{P}(b \mid desconhecido, P_{1,3}, franja)$ é igual a 1 quando a franja é consistente com as observações de brisa e é igual a 0 em caso contrário. Desse modo, para cada valor de $P_{1,3}$, efetuamos o somatório sobre os *modelos* lógicos para as variáveis de franja que são consistentes com os fatos conhecidos (compare com a enumeração sobre modelos ilustrada na Figura 7.5). Os modelos e suas probabilidades *a priori* associadas — $P(fronteira)$ — são mostrados na Figura 13.6. Temos:

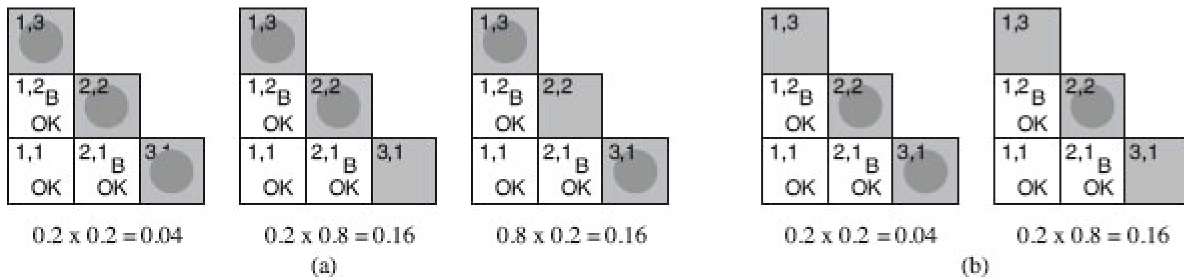


Figura 13.6 Modelos consistentes para as variáveis de fronteira $P_{2,2}$ e $P_{3,1}$, mostrando P(fronteira) para cada modelo: (a) três modelos com $P_{1,3}$ = *verdadeiro* mostrando dois ou três poços e (b) dois modelos com $P_{1,3}$ = *falso* mostrando um ou dois poços.

$$\mathbf{P}(P_{1,3} \mid conhecido, b) = \text{a}' \langle 0{,}2(0{,}04 + 0{,}16 + 0{,}16), 0{,}8(0{,}04 + 0{,}16)\rangle \approx \langle 0{,}31, 0{,}69\rangle.$$

Isto é, [1,3] (e [3,1], por simetria) contém um poço com aproximadamente 31% de probabilidade. Um cálculo semelhante, que o leitor talvez deseje realizar, mostra que [2,2] contém um poço com aproximadamente 86% de probabilidade. Definitivamente, o agente de wumpus deve evitar [2,2]! Observe que o nosso agente lógico do Capítulo 7 não sabia que [2,2] era pior do que os outros quadrados. A lógica pode nos informar que não se sabe se existe um poço em [2, 2], mas precisa da probabilidade para nos informar isso.

Esta seção mostrou que até mesmo problemas aparentemente complicados podem ser formulados com precisão em teoria da probabilidade e resolvidos com a utilização de algoritmos simples. Para se obterem soluções **eficientes**, os relacionamentos de independência e de independência

condicional podem ser empregados com a finalidade de simplificar os somatórios exigidos. Com frequência, esses relacionamentos correspondem à nossa compreensão natural de como o problema deve ser decomposto. No próximo capítulo, desenvolveremos representações formais para tais relações, bem como algoritmos que operam sobre essas representações para executar a inferência probabilística de modo eficiente.

13.7 RESUMO

Este capítulo sugeriu que a teoria da probabilidade é uma base adequada para o raciocínio incerto e forneceu uma introdução suave à sua utilização.

- A incerteza surge como consequência da preguiça e da ignorância. Ela é inevitável em mundos complexos, dinâmicos ou inacessíveis.
- As probabilidades expressam a inabilidade do agente para alcançar uma decisão definida com relação à verdade de uma sentença. As probabilidades resumem as crenças do agente relativas à evidência.
- A teoria da decisão combina as crenças do agente e desejos, definindo a melhor ação como a que maximiza a utilidade esperada.
- As declarações básicas de probabilidade incluem **probabilidades *a priori*** e **probabilidades condicionais** sobre proposições simples e complexas.
- Os axiomas da probabilidade restringem as atribuições possíveis de probabilidades a proposições. Um agente que viola os axiomas deve se comportar irracionalmente em alguns casos.
- A **distribuição de probabilidade conjunta total** especifica a probabilidade de cada atribuição completa de valores a variáveis aleatórias. Em geral, ela é grande demais para ser criada ou utilizada em sua forma explícita, mas quando está disponível pode ser utilizada para responder a consultas simplesmente adicionando entradas para os mundos possíveis correspondentes às proposições de consulta.
- A **independência absoluta** entre subconjuntos de variáveis aleatórias permitiu que a distribuição conjunta total fosse fatorada em distribuições conjuntas menores, reduzindo a sua complexidade. A independência absoluta raramente ocorre na prática.
- A **regra de Bayes** permite que probabilidades desconhecidas sejam calculadas a partir de probabilidades condicionais conhecidas, em geral no sentido causal. Normalmente, a aplicação da regra de Bayes com muitas peças de evidência resultará nos mesmos problemas de ampliação da escala que encontramos na distribuição conjunta total.
- A **independência condicional** provocada por relacionamentos causais diretos no domínio poderia permitir a fatoração da distribuição conjunta total em distribuições condicionais menores. O modelo de **Bayes ingênuo** pressupõe a independência condicional de todas as variáveis de efeito, dada uma única variável de causa, e cresce de forma linear com o número de efeitos.
- Um agente de mundo de wumpus pode calcular probabilidades relativas a aspectos não observados do mundo, melhorando assim as decisões de um agente puramente lógico. A

independência condicional torna esses cálculos tratáveis.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

A teoria da probabilidade foi inventada como uma forma de analisar os jogos de azar. Em 850 d.C., o matemático indiano Mahaviracarya descreveu como organizar um conjunto de apostas para não perder (o que hoje chamamos de livro holandês). Na Europa, as primeiras análises sistemáticas significativas foram produzidas por Girolamo Cardano, por volta de 1565, embora de publicação póstuma (1663). Nessa época, a probabilidade foi estabelecida como disciplina matemática, devido a uma série de resultados estabelecidos em uma famosa correspondência entre Blaise Pascal e Pierre de Fermat em 1654. Tal como acontece com a probabilidade em si, os resultados foram motivados inicialmente por problemas de jogo (veja o Exercício 13.9). O primeiro livro publicado sobre a probabilidade foi *De Ratiociniis in Ludo Aleae* (Huygens, 1657). A visão da incerteza "da preguiça e da ignorância" foi descrita por John Arbuthnot no prefácio de sua tradução de Huygens (Arbuthnot, 1692): "É impossível para um dado, com força e direção determinada, não cair de um lado determinado, só não conheço a força e a direção que o faz cair de tal lado e, portanto, eu chamo isso acaso, que nada mais é do que a falta de arte…"

Laplace (1816) forneceu uma visão geral excepcionalmente precisa e moderna de probabilidade, sendo o primeiro a utilizar o exemplo "tomar duas urnas, A e B, a primeira contendo quatro bolas brancas e duas bolas pretas…" O reverendo Thomas Bayes (1702-1761) introduziu a regra de raciocínio sobre probabilidades condicionais que foi denominada em homenagem a Bayes (1763). Bayes considerou apenas o caso uniforme, *a priori*, e foi Laplace quem desenvolveu independentemente o caso geral. Kolmogorov (1950, publicado pela primeira vez em alemão em 1933) apresentou pela primeira vez a teoria da probabilidade em um quadro rigorosamente axiomático. Rényi (1970), mais tarde, forneceu uma apresentação axiomática que considerou a probabilidade condicional, em vez da probabilidade absoluta, como primitiva.

Pascal utilizou a probabilidade de formas que exigiam tanto a interpretação objetiva como uma propriedade do mundo baseada em simetria ou frequência relativa, quanto à interpretação subjetiva, baseada no grau de crença — a primeira em suas análises de probabilidades em jogos de azar, e a outra no famoso argumento "aposta de Pascal" sobre a possível existência de Deus. Porém, Pascal não percebeu claramente a distinção entre essas duas interpretações. A distinção foi primeiro percebida de maneira clara por James Bernoulli (1654-1705).

Leibniz introduziu a noção "clássica" de probabilidade como uma proporção de casos enumerados igualmente prováveis, que também foi utilizada por Bernoulli, embora tenha alcançado proeminência graças a Laplace (1749-1827). Essa noção é ambígua entre a interpretação de frequência e a interpretação subjetiva. Os casos podem ser considerados igualmente prováveis devido a uma simetria física natural entre eles ou simplesmente porque não temos qualquer conhecimento que nos leve a considerar um deles mais provável que o outro. O uso desta última consideração subjetiva para justificar a atribuição de probabilidades iguais é conhecido como **princípio da indiferença**. O princípio é frequentemente atribuído a Laplace, mas ele nunca o isolou explicitamente. George Boole e John Venn referiram-se a ele como **princípio da insuficiência da razão**; o nome moderno deve-se a

Keynes (1921).

O debate entre objetivistas e subjetivistas se tornou mais acirrado no século XX. Kolmogorov (1963), R. A. Fisher (1922) e Richard von Mises (1928) defenderam a interpretação de frequência relativa. A interpretação de "propensão" de Karl Popper (1959, publicada primeiro em alemão em 1934) segue as frequências relativas até uma simetria física subjacente. Frank Ramsey (1931), Bruno de Finetti (1937), R. T. Cox (1946), Leonard Savage (1954), Richard Jeffrey (1983) e E. T. Jaynes (2003) interpretaram as probabilidades como graus de crença de indivíduos específicos. Suas análises de grau de crença estavam intimamente ligadas a utilidades e a comportamento — especificamente, à disposição para apostar. Rudolf Carnap, seguindo o trabalho de Leibniz e Laplace, ofereceu uma espécie diferente de interpretação subjetiva da probabilidade — não como o grau de crença de qualquer indivíduo real, mas como o grau de crença que um indivíduo *idealizado* deve ter em determinada proposição a, dado um corpo de evidências específico $\mathbf{e}$. Carnap tentou ir além de Leibniz ou Laplace, tornando essa noção de grau de **confirmação** matematicamente precisa, como uma relação lógica entre a e $\mathbf{e}$. O estudo dessa relação foi planejado para constituir uma disciplina matemática chamada **lógica indutiva**, análoga à lógica dedutiva comum (Carnap, 1948, 1950). Carnap não foi capaz de estender sua lógica indutiva muito além do caso proposicional, e Putnam (1963) mostrou que algumas dificuldades fundamentais impediriam uma extensão rígida a linguagens capazes de expressar a aritmética.

O teorema de Cox (1946) mostra que qualquer sistema de raciocínio incerto que atenda seu conjunto de hipóteses é equivalente à teoria da probabilidade. Isso deu confiança renovada aos que já favoreciam a probabilidade, mas outros não estavam convencidos, apontando para os pressupostos (principalmente que a crença deve ser representada por um único número e, portanto, a crença em $\neg p$ deve ser uma função da crença em p). Halpern (1999) descreveu as premissas e mostrou algumas lacunas na formulação original de Cox. Horn (2003) mostrou como remediar as dificuldades. Jaynes (2003) tinha um argumento semelhante que é mais fácil de ler.

A questão de classes de referência está estreitamente ligada à tentativa de descobrir uma lógica indutiva. A abordagem de escolher a classe de referência "mais específica" de tamanho suficiente foi proposta formalmente por Reichenbach (1949). Várias tentativas foram feitas, notavelmente por Henry Kyburg (1977, 1983), para formular normas mais sofisticadas com a finalidade de evitar algumas falácias óbvias que surgem com a regra de Reichenbach, mas tais abordagens permanecem um tanto *ad hoc*. O trabalho mais recente de Bacchus, Grove, Halpern e Koller (1992) estende os métodos de Carnap às teorias de primeira ordem, evitando assim muitas das dificuldades associadas com o método direto de classe de referência. Kyburg e Teng (2006) contrastaram a inferência probabilística com a lógica não monotônica.

O raciocínio probabilístico de Bayes foi usado em IA desde a década de 1960, especialmente em diagnóstico médico. Ele foi empregado não apenas para fazer um diagnóstico a partir da evidência disponível, mas também para selecionar questões e testes adicionais utilizando a teoria do valor da informação (Seção 16.6) quando a evidência disponível era inconclusiva (Gorry, 1968; Gorry *et al.*, 1973). Um sistema superou os especialistas humanos no diagnóstico de enfermidades abdominais agudas (Dombal *et al.*, 1974). Lucas *et al.* (2004) apresentou um panorama. Entretanto, esses primeiros sistemas de Bayes se ressentiam de diversos problemas. Pelo fato de não terem qualquer modelo teórico das condições em que estavam efetuando os diagnósticos, eles eram vulneráveis a

dados não representativos que ocorrem em situações nas quais somente uma pequena amostra estava disponível (Dombal *et al.*, 1981). Um problema ainda mais fundamental surgiu porque lhes faltavam um formalismo conciso (como o que descrevemos no Capítulo 14) para representar e utilizar informações de independência condicional e, por essa razão, eles dependiam da aquisição, do armazenamento e do processamento de enormes tabelas de dados probabilísticos. Devido a essas dificuldades, os métodos probabilísticos para lidar com a incerteza foram os preferidos em IA, desde a década de 1970 até a metade da década de 1980. Os desenvolvimentos ocorridos no final dos anos 1980 serão descritos no próximo capítulo.

O modelo de Bayes ingênuo para distribuições conjuntas foi extensivamente estudado na literatura de reconhecimento de padrões desde a década de 1950 (Duda e Hart, 1973). Também foi utlizado, muitas vezes de forma inconsciente, na recuperação de informação, começando com o trabalho de Maron (1961). Os fundamentos probabilísticos dessa técnica, descrita mais adiante no Exercício 13.22, foram elucidados por Robertson e Sparck Jones (1976). Domingos e Pazzani (1997) apresentam uma explicação para o surpreendente sucesso do raciocínio de Bayes ingênuo, até mesmo em domínios nos quais as hipóteses de independência são claramente violadas.

Existem muitos livros didáticos introdutórios de boa qualidade sobre teoria da probabilidade, inclusive os de Bertsekas e Tsitsiklis (2008) e Grinstead e Snell (1997). DeGroot e Schervish (2001) oferece uma introdução combinada à probabilidade e à estatística de um ponto de vista bayesiano. O livro didático de Richard Hamming (1991) fornece uma introdução matematicamente sofisticada à teoria da probabilidade, sob o ponto de vista de uma interpretação de propensão baseada na simetria física. Hacking (1975) e Hald (1990) focalizam a história inicial do conceito de probabilidade. Bernstein (1996) apresenta uma interessante história popular sobre o risco.

EXERCÍCIOS

13.1 Mostre, a partir de princípios básicos, que $P(a \mid b \wedge a) = 1$.

13.2 Usando os axiomas de probabilidade, prove que qualquer distribuição de probabilidade sobre uma variável aleatória discreta deve ter a soma 1.

13.3 Para cada uma das seguintes afirmações, prove que é verdade ou dê um contraexemplo.

a. Se $P(a \mid b, c) = P(b \mid a, c)$, então $P(a \mid c) = P(b \mid c)$

b. Se $P(a \mid b, c) = P(a)$, então $P(b \mid c) = P(b)$

c. Se $P(a \mid b) = P(a)$, então $P(a \mid b, c) = P(a \mid c)$

13.4 Seria racional para um agente conter as três crenças $P(A) = 0{,}4$, $P(B) = 0{,}3$ e $P(A \vee B) = 0{,}5$? Nesse caso, que intervalo de probabilidades seria racional o agente conter para $A \wedge B$? Componha uma tabela semelhante à da Figura 13.2 e mostre como ela apoia seu argumento sobre a racionalidade. Em seguida, elabore outra versão da tabela, onde $P(A \vee B) = 0{,}7$. Explique por que é racional ter essa probabilidade, embora a tabela mostre um caso de perda e três que simplesmente indicam equilíbrio. (*Sugestão*: Qual é o compromisso do Agente 1 com a probabilidade de cada um dos quatro casos, em especial com o caso que representa uma perda?)

13.5 Esta questão lida com as propriedades dos mundos possíveis, definida como atribuições para todas as variáveis aleatórias. Vamos trabalhar com proposições que correspondam a exatamente um mundo possível porque ela compromete as atribuições de todas as variáveis. Em teoria da probabilidade, tais proposições são chamados de **eventos atômicos**. Por exemplo, com as variáveis booleanas X_1, X_2, X_3, a proposição $x_1 \wedge \neg x_2 \wedge \neg x_3$ fixa a atribuição das variáveis; na linguagem da lógica proposicional, diríamos que ela tem exatamente um modelo.

a. Prove, para o caso de n variáveis booleanas, que quaisquer dois eventos distintos atômicos são mutuamente exclusivos, ou seja, sua conjunção é equivalente a *falso*.

b. Prove que a disjunção de todos os eventos atômicos possíveis é logicamente equivalente a *verdadeiro*.

c. Prove que qualquer proposição é logicamente equivalente à disjunção dos eventos atômicos que impõem sua verdade.

13.6 Prove a Equação 13.4 pelas Equações 13.1 e 13.2.

13.7 Considere o conjunto de cinco cartas possíveis da mão do jogo de pôquer tratadas de forma honesta de um baralho-padrão de 52 cartas.

a. Quantos eventos atômicos existem na distribuição de probabilidade conjunta (isto é, quantas mãos de cinco cartas existem)?

b. Qual é a probabilidade de cada evento atômico?

c. Qual é a probabilidade de ser distribuído um *royal straight flush*? E quatro cartas de um mesmo naipe?

13.8 Dada a distribuição conjunta total mostrada na Figura 13.3, calcule:

a. $\mathbf{P}(dordedente.)$

b. $\mathbf{P}(Cárie)$.

c. $\mathbf{P}(DorDeDente \mid cárie)$.

d. $\mathbf{P}(Cárie \mid dordedente \vee Boticão)$.

13.9 Em sua carta de 24 de agosto de 1654, Pascal estava tentando mostrar como um pote de dinheiro deveria ser distribuído quando um jogo de apostas terminasse prematuramente. Imagine um jogo que consista no lançamento de um dado de cada vez, o jogador E ganha um ponto quando o dado for par, e o jogador O ganha um ponto quando o dado for ímpar. O primeiro jogador a receber sete pontos ganha o pote. Suponha que o jogo tenha sido interrompido quando estava 4×2 para E. Como o dinheiro deverá ser dividido de forma justa nesse caso? Qual é a fórmula geral? (Fermat e Pascal cometeram vários erros antes de resolver o problema, mas você deverá ser capaz de acertar na primeira vez.)

13.10 Ao decidir colocar nosso conhecimento de probabilidade para bom uso, deparamo-nos com uma máquina caça-níqueis com três bobinas girando de forma independente, cada uma produzindo um dos quatro símbolos, BAR, SINO, LIMÃO ou CEREJA com igual probabilidade. A máquina caça-níqueis tem o seguinte esquema de pagamento para aposta com uma moeda (onde "?" indica que não interessa o que aparece naquela bobina):

BAR/ BAR/BAR paga 20 moedas

SINO/SINO/SINO paga 15 moedas

LIMÃO/LIMÃO/LIMÃO paga 5 moedas

CEREJA/CEREJA/CEREJA paga 3 moedas

CEREJA/CEREJA/? paga 2 moedas

CEREJA/? /? paga 1 moeda

a. Calcule a porcentagem de “recuperação de investimento” esperado pela máquina. Em outras palavras, para cada moeda jogada, qual o retorno de moeda esperado?

b. Calcule a probabilidade de uma única jogada na máquina caça-níqueis resultar em vitória.

c. Estime o número de jogadas média e mediana que você pode esperar fazer até quebrar, se começar com oito moedas. Você pode executar uma simulação para estimar isso, em vez de tentar calcular a resposta exata.

13.11 Desejamos transmitir uma mensagem de n bits para um agente receptor. Os bits da mensagem podem ser corrompidos (invertidos) independentemente durante a transmissão, com $\in$ probabilidade cada. Com um bit de paridade extra enviado junto com a informação original, uma mensagem pode ser corrigida pelo receptor se, no máximo, um bit na mensagem inteira (incluindo o bit de paridade) foi corrompido. Suponha que queiramos garantir que a mensagem correta seja recebida com probabilidade pelo menos $1 - \delta$. Qual o valor máximo possível de n? Calcule esse valor para o caso $\in = 0{,}002$, $\delta = 0{,}01$.

13.12 Mostre que as três formas de independência na Equação 13.11 são equivalentes.

13.13 Considere dois testes médicos, A e B, de um vírus. O teste A tem 95% de chance de reconhecer o vírus quando estiver presente, mas tem uma proporção de 10% de falso positivo (indicando que o vírus está presente quando não está). O teste B é 90% eficaz em reconhecer o vírus, mas tem uma proporção de 5% de falso positivo. Os dois testes utilizam métodos independentes de identificação do vírus. Um por cento de todas as pessoas são portadoras do vírus. Digamos que utilizemos em uma pessoa apenas um dos testes para detectar o vírus e que o teste resulte positivo para portadores do vírus. Qual dos dois testes, retornando positivo, é mais indicativo de que alguém seja realmente portador do vírus? Justifique sua resposta matematicamente.

13.14 Suponha que seja dada uma moeda com a probabilidade x de dar cara e a probabilidade $1-x$ de dar coroa. Os resultados dos arremessos sucessivos da moeda são independentes um do outro se você conhecer o valor de x? Os resultados dos arremessos sucessivos da moeda são independentes um do outro se você não conhecer o valor de x? Justifique sua resposta.

13.15 Depois de seu *checkup* anual, o médico tem notícias boas e ruins. As notícias ruins são que você teve um resultado positivo no exame referente a uma doença séria e que o exame é 99% preciso (isto é, a probabilidade de o exame ser positivo quando você tem a doença é 0,99, pois essa é a probabilidade de resultado negativo no exame quando você não tem a doença). A boa notícia é que essa é uma doença rara, atingindo apenas uma em 10.000 pessoas da sua idade. Por que é uma boa notícia o fato de a doença ser rara? Quais são suas chances de ter realmente a doença?

13.16 Com muita frequência, é útil considerar o efeito de algumas proposições específicas no

contexto de alguma evidência prática geral que permanece fixa, em vez de considerá-lo na ausência completa de informações. As perguntas a seguir lhe pedem para provar versões mais gerais da regra do produto e da regra de Bayes, em relação a alguma evidência prática **e**:

a. Prove a versão condicionalizada da regra do produto geral:

$$\mathbf{P}(X, Y \mid \mathbf{e}) = \mathbf{P}(X \mid Y, \mathbf{e})\mathbf{P}\,(Y \mid \mathbf{e}).$$

b. Prove a versão condicionalizada da regra de Bayes na Equação 13.13.

13.17 Mostre que a declaração de independência condicional

$$\mathbf{P}(X, Y \mid Z) = \mathbf{P}(X \mid Z)\mathbf{P}(Y \mid Z)$$

é equivalente a qualquer das duas declarações

$$\mathbf{P}(X \mid Y,Z) = \mathbf{P}(X \mid Z) \text{ e } \mathbf{P}(B \mid X,Z) = \mathbf{P}(Y \mid Z).$$

13.18 Suponha que você receba uma sacola com n moedas imparciais. Das moedas, n – 1 são normais, com cara de um lado e coroa de outro, enquanto que uma delas é falsa, com cara em ambos os lados.

a. Suponha que você tire de dentro da sacola uma moeda aleatoriamente, arremesse e apareça cara. Qual a probabilidade (condicional) que a moeda escolhida seja a falsa?

b. Suponha que você continue a arremessar a moeda **K** vezes após apanhá-la e aparece k caras. Qual é agora a probabilidade condicional de pegar uma moeda falsa?

c. Suponha que você gostaria de decidir se a moeda escolhida era a falsa ao arremessa-la k vezes. O procedimento de decisão retorna *falso* se todos os k arremessos apresentam caras; de outra forma retorna *normal*. Qual é a probabilidade (incondicional) que esse procedimento tenha um erro?

13.19 Neste exercício, você completará o cálculo de normalização para o exemplo da meningite. Primeiro, componha um valor apropriado para $P(s|\neg m)$ e use esse valor para calcular valores não normalizados para $P(m|s)$ e $P(\neg m|s)$ [isto é, ignorando o termo $P(s)$ na expressão da regra de Bayes (equação 13.14)]. Depois disso, normalize esses valores de forma que eles totalizem 1.

13.20 Sejam X, Y, Z variáveis booleanas aleatórias. Identifique as oito entradas na distribuição conjunta $\mathbf{P}(X, Y, Z)$ com as letras de a até h. Expresse, sob a forma de um conjunto de equações relacionadas às entradas de a até h, a declaração de que X e Y são condicionalmente independentes, dada a variável Z. Quantas equações *não redundantes* existem nesse conjunto?

13.21 (Adaptado de Pearl (1988).) Suponha que você seja testemunha de um acidente noturno seguido de fuga envolvendo um taxi em Atenas. Todos os taxis em Atenas são azuis ou cinza. Você jura de pés juntos que o taxi era azul. Testes extensivos mostram que, em condição de luz fraca, a distinção entre cinza e azul é 75% confiável.

a. É possível calcular a cor mais provável do taxi?(*Dica*: mostre cuidadosamente a diferença entre a proposição que o taxi seja azul e a proposição que *pareça* azul.)

b. E se você soubesse que 9 entre 10 taxis em Atenas são verdes?

13.22 Categorizar um texto é a tarefa de atribuir a determinado documento uma categoria de um conjunto fixo de categorias, com base no texto que o documento contém. Os modelos de Bayes

ingênuos são empregados com frequência para essa tarefa. Nesses modelos, a variável de consulta é a categoria do documento, e as variáveis de "efeito" são a presença ou ausência de cada palavra na linguagem; a suposição é que as palavras ocorrem independentemente nos documentos, com suas respectivas frequências determinadas pela categoria de cada documento.

a. Explique precisamente como tal modelo pode ser construído, sendo fornecido para servir como "dados de treinamento" um conjunto de documentos que foram atribuídos a categorias.

b. Explique precisamente como categorizar um novo documento.

c. A suposição de independência é razoável? Explique.

13.23 Em nossa análise do mundo de wumpus, usamos o fato de que cada quadrado contém um poço com probabilidade 0,2, independentemente do conteúdo dos outros quadrados. Suponha que, em vez disso, exatamente $N/5$ poços estejam uniformemente espalhados ao acaso entre os N quadrados diferentes de [1,1]. As variáveis $P_{i,j}$ e $P_{k,l}$ ainda serão independentes? Qual será agora a distribuição conjunta $\mathbf{P}(P_{1,1}, \ldots, P_{4,4})$? Efetue novamente os cálculos para as probabilidades de poços em [1,3] e [2,2].

13.24 Refaça o cálculo da probabilidade dos poços em [1,3] e [2,2] assumindo que cada quadrado contém um poço com probabilidade 0,01, independente dos outros quadrados. O que você pode dizer sobre o desempenho relativo de uma lógica *versus* um agente probabilístico nesse caso?

13.25 Implemente um agente probabilístico híbrido para o mundo de wumpus, baseado no agente híbrido da Figura 7.20 e no procedimento de inferência probabilística descrito neste capítulo.

---

[1] Por ora, assumiremos um conjunto de mundos discreto, contável. O tratamento adequado do caso contínuo traz certas complicações que são menos relevantes para a maioria dos propósitos em IA.

[2] As dificuldades incluem o **conjunto de Vitali**, um conjunto bem definido do intervalo [0, 1] sem tamanho bem definido.

[3] Pode-se argumentar que as preferências do agente por balanços financeiros diferentes são tais que a possibilidade de perder $1 não é contrabalançada pela igual possibilidade de ganhar $1. Uma resposta possível é fazer com que o valor das apostas diminua o suficiente para evitar esse problema. A análise de Savage (1954) contorna o problema completamente.

[4] Assim chamada devido a uma prática comum entre as companhias de seguros de escrever as somas das frequências observadas nas margens de tabelas de seguros.

[5] Supomos que o paciente e o dentista sejam indivíduos distintos.

CAPÍTULO 14

Raciocínio probabilístico

Em que explicamos como construir modelos de redes para raciocinar sob a incerteza, de acordo com as leis de teoria da probabilidade.

O Capítulo 13 apresentou os elementos básicos da teoria da probabilidade e observou a importância dos relacionamentos de independência e de independência condicional na simplificação de representações probabilísticas do mundo. Este capítulo introduz um modo sistemático de representar explicitamente tais relacionamentos, sob a forma de **redes bayesianas**. Definimos a sintaxe e a semântica dessas redes e mostramos como elas podem ser usadas para captar o conhecimento incerto de modo natural e eficiente. Em seguida, mostramos como a inferência probabilística, embora computacionalmente intratável no pior caso, pode ser realizada de maneira eficiente em muitas situações práticas. Também descrevemos uma variedade de algoritmos de inferência aproximados, frequentemente aplicáveis quando a inferência exata é inviável. Exploramos modos de aplicar a teoria da probabilidade a mundos com objetos e relações, isto é, a *representações de primeira ordem*, em vez de *proposicionais*. Por fim, estudamos abordagens alternativas para o raciocínio incerto.

14.1 REPRESENTAÇÃO DO CONHECIMENTO EM UM DOMÍNIO INCERTO

No Capítulo 13, vimos que a distribuição de probabilidade conjunta total pode responder a qualquer pergunta sobre o domínio, mas pode se tornar intratavelmente grande, à medida que o número de variáveis cresce. Além disso, especificar probabilidades para mundos possíveis, uma por uma, é antinatural e tedioso.

Também vimos que os relacionamentos de independência e de independência condicional entre variáveis pode reduzir bastante o número de probabilidades que precisam ser especificadas, a fim de definir a distribuição conjunta total. Esta seção introduz uma estrutura de dados chamada **rede bayesiana**[1] para representar as dependências entre variáveis. As redes bayesianas podem representar essencialmente qualquer distribuição de probabilidade conjunta completa e, em muitos casos, muito concisamente.

Uma rede bayesiana é um grafo orientado em que cada *nó* é identificado com informações de

probabilidade quantitativa. A especificação completa é dada a seguir:

1. Cada nó corresponde a uma variável aleatória, que pode ser discreta ou contínua.
2. Um conjunto de vínculos orientados ou setas conecta pares de nós. Se houver uma seta do nó X até o nó Y, X será denominado *pai* de Y. O grafo não tem ciclos orientados (e, portanto, é um grafo acíclico orientado, ou GAO).
3. Cada nó X_i tem uma distribuição de probabilidade condicional $\mathbf{P}(X_i \mid Pais(X_i))$ que quantifica o efeito dos pais sobre o nó.

A topologia da rede — o conjunto de nós e vínculos — especifica os relacionamentos de independência condicional que são válidos no domínio, de um modo que se tornará claro em breve. O significado *intuitivo* de uma seta tipicamente é que X tem *influência direta* sobre Y, o que sugere que as causas devem ser pais dos efeitos. Normalmente é fácil para um especialista em domínios descobrir quais são as influências diretas existentes no domínio — na verdade, é muito mais fácil do que realmente especificar as probabilidades em si. Uma vez que a topologia da rede bayesiana é definida, só precisamos especificar uma distribuição de probabilidade condicional para cada variável, dados seus pais. Veremos que a combinação da topologia com as distribuições condicionais basta para especificar (de forma implícita) a distribuição conjunta total para todas as variáveis.

Lembre-se do mundo simples descrito no Capítulo 13, que consiste nas variáveis *DorDeDente*, *Cárie*, *Boticão* e *Tempo*. Argumentamos que *Tempo* é independente das outras variáveis; além disso, observamos que *DorDeDente* e *Boticão* são condicionalmente independentes, dada *Cárie*. Esses relacionamentos são representados pela estrutura de rede bayesiana mostrada na Figura 14.1. Formalmente, a independência condicional de *DorDeDente* e *Boticão* dada *Cárie* é a *ausência* de um vínculo entre *DorDeDente* e *Boticão*. Intuitivamente, a rede representa o fato de que *Cárie* é uma causa direta de *DorDeDente* e *Boticão*, enquanto não existe nenhum relacionamento causal direto entre *DorDeDente* e *Boticão*.

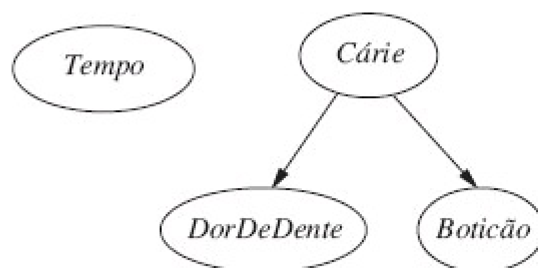


Figura 14.1 Uma rede bayesiana simples, na qual *Tempo* é independente das outras três variáveis, e *DorDeDente* e *Boticão* são condicionalmente independentes, dada *Cárie*.

Agora considere o exemplo a seguir, que é apenas um pouco mais complexo. Você tem um novo alarme contra assaltantes instalado em sua casa. Ele é bastante confiável na detecção de um roubo, mas também responde ocasionalmente a pequenos terremotos. (Esse exemplo se deve a Judea Pearl, residente em Los Angeles — daí o interesse em terremotos.) Você também tem dois vizinhos, João e Maria, que prometeram chamá-lo no trabalho quando ouvirem o alarme. João quase sempre chama quando ouve o alarme, mas às vezes confunde o toque do telefone com o alarme e também liga ao

ouvi-lo. Por outro lado, Maria gosta de ouvir música em alto volume e frequentemente esquece completamente o alarme. Dada a evidência de quem telefonou ou não telefonou, gostaríamos de estimar a probabilidade de um roubo.

A rede bayesiana para esse domínio é dada na Figura 14.2. A estrutura da rede mostra que o roubo e os terremotos afetam diretamente a probabilidade de o alarme disparar se o telefonema de João e Maria depender apenas do alarme. A rede representa, portanto, nossas suposições de que eles não percebem roubos diretamente, não notam pequenos terremotos e não conferem antes de telefonar.

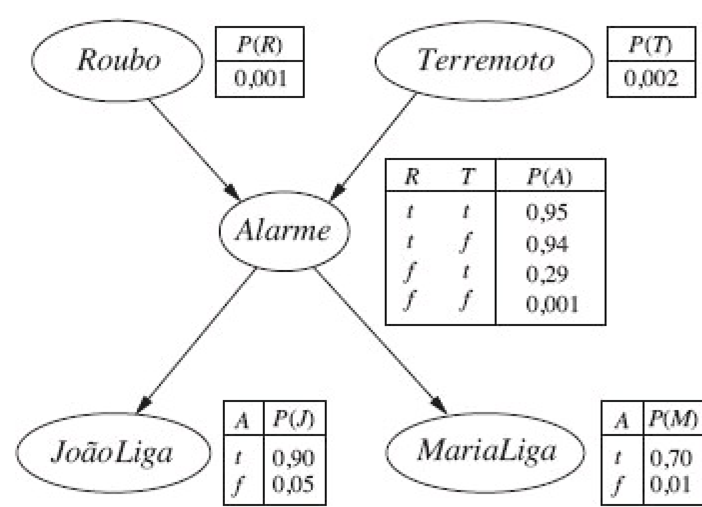


Figura 14.2 Uma rede bayesiana típica, mostrando a topologia e também as tabelas de probabilidade condicional (TPCs). Nas TPCs, as letras R, T, A, J e M representam *Roubo*, *Terremoto*, *Alarme*, *JoãoLiga* e *MariaLiga*, respectivamente.

As distribuições condicionais na Figura 14.2 são mostradas como uma **tabela de probabilidade condicional**, ou TPC (essa forma de tabela pode ser usada para variáveis discretas; outras representações incluem as adequadas às variáveis contínuas, descritas na Seção 14.2). Cada linha da TPC contém a probabilidade condicional de cada valor do nó para um **caso de condicionamento**. Um caso de condicionamento é apenas uma combinação possível de valores para os nós pai — uma miniatura do mundo possível. Cada linha deve somar 1 porque as entradas representam um conjunto exaustivo de casos da variável. Para as variáveis booleanas, uma vez que se sabe que a probabilidade de um valor verdadeiro seja p, a probabilidade de falso deve ser $1 - p$; assim, muitas vezes omitimos o segundo número, como na Figura 14.2. Em geral, uma tabela para uma variável booleana com k pais booleanos contém 2^k probabilidades independentemente especificáveis. Um nó sem pais tem apenas uma linha representando as probabilidades anteriores a cada valor possível da variável.

Note que a rede não tem nós correspondentes ao fato de Maria estar ouvindo música em alto volume no momento ou ao fato de o telefone tocar e confundir João. Esses fatores são resumidos na incerteza associada aos vínculos de *Alarme* para *JoãoLiga* e *MariaLiga*. Isso mostra ao mesmo tempo a preguiça e a ignorância em operação: seria muito trabalhoso descobrir por que esses fatores seriam mais ou menos prováveis em qualquer caso específico e, na verdade, não temos nenhum modo razoável de obter as informações relevantes. As probabilidades realmente resumem um conjunto *potencialmente infinito* de circunstâncias em que o alarme poderia deixar de soar (umidade elevada, falta de energia, bateria descarregada, fios cortados, um rato morto preso à campainha etc.) ou então João ou Maria podem deixar de ligar para informar que ele soou (saíram para almoçar, saíram de

férias, estão temporariamente surdos, está passando um helicóptero etc.). Desse modo, um pequeno agente pode lidar com um mundo muito grande, pelo menos aproximadamente. O grau de aproximação pode ser melhorado se introduzirmos informações relevantes adicionais.

14.2 A SEMÂNTICA DAS REDES BAYESIANAS

A seção anterior descreveu o que é uma rede, mas não o que ela significa. Há duas maneiras de compreender a semântica das redes bayesianas. A primeira é ver a rede como uma representação da distribuição de probabilidade conjunta. A segunda é visualizá-la como uma codificação de uma coleção de declarações de independência condicional. As duas visões são equivalentes, mas a primeira se mostra útil na compreensão de como *construir* redes, enquanto a segunda é útil no projeto de procedimentos de inferência.

14.2.1 Representação da distribuição conjunta total

Visto como um pedaço de "sintaxe", uma rede bayesiana é um grafo acíclico orientado com alguns parâmetros numéricos ligados a cada nó. Uma maneira de definir o que significa a rede — sua semântica — é definir a maneira pela qual ela representa uma distribuição conjunta específica sobre todas as variáveis. Para fazer isso, precisamos primeiro retirar (temporariamente) o que foi dito anteriormente sobre os parâmetros associados a cada nó. Dissemos que esses parâmetros correspondem às probabilidades condicionais $\mathbf{P}(X_i \mid Pais\ (X_i))$; essa é uma afirmação verdadeira, mas até atribuirmos a semântica à rede como um todo devemos considerá-los apenas como números $\theta(X_i \mid Pais\ (X_i))$.

Uma entrada genérica na distribuição conjunta é a probabilidade de uma conjunção de atribuições específicas a cada variável, tal como $P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n)$. Usamos a notação $P(x_1, \ldots, x_n)$ como abreviação para isso. O valor dessa entrada é dado pela fórmula:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} \theta(x_i \mid pais\,(X_i)) \,, \tag{14.1}$$

onde $pais(X_i)$ denota os valores em $Pais(X_i)$ que aparece em $x_1,\ldots,x_n$. Desse modo, cada entrada na distribuição conjunta é representada pelo produto dos elementos apropriados das tabelas de probabilidade condicional (TPCs) na rede bayesiana.

A partir dessa definição, é fácil provar que os parâmetros $\theta(X_i \mid Pais\ (X_i))$ são exatamente as probabilidades condicionais $\mathbf{P}(X_i \mid Pais\ (X_i))$ deduzidas pela distribuição conjunta (ver Exercício 14.2). Assim, podemos reescrever a Equação 14.1 como

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid pais\,(X_i)) \,. \tag{14.2}$$

Em outras palavras, as tabelas que chamamos de tabelas de probabilidade condicional realmente *são* tabelas de probabilidade condicional de acordo com a semântica definida na Equação 14.1.

Para ilustrar isso, podemos calcular a probabilidade de que o alarme tenha soado, mas não tenha ocorrido nenhum roubo nem terremoto, e que tanto João quanto Maria tenham ligado. Multiplicamos as entradas da distribuição conjunta (usando nomes de letras únicas para as variáveis):

$$P(j, m, a, \neg b, \neg e) = P(j \mid a)P(m \mid a)P(a \mid \neg b \wedge \neg e)P(\neg b)P(\neg e)$$
$$= 0{,}90 \times 0{,}70 \times 0{,}001 \times 0{,}999 \times 0{,}998 = 0{,}000628.$$

A Seção 13.3 explicou que a distribuição conjunta total pode ser usada para responder a qualquer consulta sobre o domínio. Se uma rede bayesiana for uma representação da distribuição conjunta, ela também poderá ser usada para responder a qualquer consulta, efetuando-se o somatório de todas as entradas conjuntas relevantes. A Seção 14.4 explica como fazer isso, mas também descreve métodos que são muito mais eficientes.

Um método para construir redes bayesianas

A Equação 14.2 define o que significa uma rede bayesiana. A próxima etapa é como *construir* uma rede bayesiana de tal modo que a distribuição conjunta resultante seja uma boa representação de dado domínio. Agora, mostraremos que a Equação 14.2 implica certos relacionamentos de independência condicional que podem ser usados para orientar o engenheiro do conhecimento na construção da topologia da rede.

Primeiro, reescrevemos as entradas na distribuição conjunta em termos de uma probabilidade condicional usando a regra do produto:

$$P(x_1, \ldots, x_n) = P(x_n | x_{n-1}, \ldots, x_1)P(x_{n-1}, \ldots, x_1).$$

Em seguida, repetimos o processo reduzindo cada probabilidade conjuntiva a uma probabilidade condicional e uma conjunção menor. Terminamos com um grande produto:

$$P(x_1, \ldots, x_n) = P(x_n | x_{n-1}, \ldots, x_1)P(x_{n-1} | x_{n-2}, \ldots, x_1) \ldots P(x_2 | x_1)P(x_1)$$
$$= \prod_{i=1}^{n} P(x_i \mid x_{i-1}, \ldots, x_1) .$$

Essa identidade é chamada de **regra da cadeia**. É válida para qualquer conjunto de variáveis aleatórias. Comparando-a com a Equação 14.2, vemos que a especificação da distribuição conjunta é equivalente à afirmação geral de que, para toda variável X_i na rede, temos

$$\mathbf{P}(X_i | X_{i-1}, \ldots, X_1) = \mathbf{P}(X_i | Pais(X_i)), \qquad (14.3)$$

desde que $Pais(X_i) \subseteq \{X_{i-1}, \ldots, X_1\}$. Esta última condição é satisfeita enumerando os nós em qualquer ordem consistente com a ordem parcial implícita na estrutura do grafo.

O que a Equação 14.3 nos diz é que a rede bayesiana é uma representação correta do domínio somente se cada nó é condicionalmente independente de seus predecessores na ordenação de nós, dados seus pais. Podemos satisfazer essa condição com esta metodologia:

1. *Nós*: Primeiro determine o conjunto de variáveis que são necessárias para modelar o domínio. Agora as ordene, $\{X_1, \ldots, X_n\}$. Qualquer ordem vai funcionar, mas a rede resultante será mais compacta se as variáveis forem ordenadas de tal forma que as causas precedam os efeitos.
2. *Vínculos*: Para $i = 1$ até n faça:
   - Escolha, de $X_1,\ldots, X_{i-1}$, um conjunto mínimo de pais para X_i, tal que a Equação 14.3 seja satisfeita.
   - Para cada pai insira um vínculo do pai para X_i.
   - TPCs: escreva a tabela de probabilidade condicional, $\mathbf{P}(X_i \mid Pais\ (X_i))$.

Intuitivamente, os pais do nó X_i devem conter todos os nós em $X_1, \ldots, X_{i-1}$ que *influenciam diretamente* X_i. Por exemplo, vamos supor que completamos a rede da Figura 14.2, exceto pela escolha de pais para *MariaLiga*. *MariaLiga* certamente é influenciada pelo fato de haver ou não um *Roubo* ou um *Terremoto*, mas não é *diretamente* influenciada. Intuitivamente, nosso conhecimento do domínio nos diz que esses eventos influenciam a disposição de Maria para telefonar somente por seu efeito sobre o alarme. Além disso, dado o estado do alarme, o fato de João ligar não tem influência sobre a ligação de Maria. Em termos formais, acreditamos que a declaração de independência condicional a seguir seja válida:

$$\mathbf{P}(MariaLiga \mid JoãoLiga, Alarme, Terremoto, Roubo) = \mathbf{P}(MariaLiga \mid Alarme).$$

Assim, *Alarme* será o único nó pai para *MariaLiga.*

Como cada nó só é ligado aos nós anteriores, esse método de construção garante que a rede é acíclica. Outra propriedade importante da rede bayesiana é que ela não contém valores de probabilidade redundante. Se não houver redundância, não há chance para inconsistência: é *impossível para o engenheiro de conhecimento ou especialista de domínio criar uma rede bayesiana que viole os axiomas da probabilidade.*

Densidade e ordenação de nós

Além de ser uma representação completa e não redundante do domínio, uma rede bayesiana frequentemente pode ser muito mais *compacta* que a distribuição conjunta total. Essa propriedade é o que torna viável manipular domínios com muitas variáveis. A densidade das redes bayesianas é um exemplo de propriedade muito geral de **sistemas localmente estruturados** (também chamados **sistemas esparsos**). Em um sistema localmente estruturado, cada subcomponente interage diretamente apenas com um número limitado de outros componentes, não importando o número total de componentes. A estrutura local normalmente está associada a um crescimento linear, e não a um crescimento exponencial da complexidade. No caso das redes bayesianas, é razoável supor que, na maioria dos domínios, cada variável aleatória é diretamente influenciada por, no máximo, k outras, para alguma constante k. Se supusermos n variáveis booleanas por simplicidade, a quantidade de informações necessárias para especificar cada tabela de probabilidade condicional será no máximo 2^k números, e a rede completa poderá ser especificada por $n2^k$ números. Em contraste, a distribuição conjunta contém 2^n números. Para tornar esse exemplo concreto, vamos supor que tenhamos $n = 30$

nós, cada um com cinco pais ($k = 5$). Então, a rede bayesiana exigirá 960 números, mas a distribuição conjunta total exigirá mais de um bilhão.

Existem domínios em que cada variável pode ser diretamente influenciada por todas as outras, de forma que a rede seja totalmente conectada. Então, a especificação das tabelas de probabilidade condicional exige a mesma quantidade de informações que a especificação da distribuição conjunta. Em alguns domínios, existirão dependências fracas que deverão ser incluídas estritamente pela adição de novos vínculos. Porém, se essas dependências forem muito tênues, talvez não compense a complexidade adicional na rede em relação ao pequeno ganho em exatidão. Por exemplo, alguém poderia fazer uma objeção à nossa rede de alarme contra roubo afirmando que, se houvesse um terremoto, João e Maria não telefonariam mesmo que tivessem ouvido o alarme porque eles iriam supor que o terremoto fosse a causa. A decisão de adicionar o vínculo de *Terremoto* para *JoãoLiga* e para *MariaLiga* (e, desse modo, de ampliar as tabelas) dependerá da comparação entre a importância de obter probabilidades mais precisas e o custo de especificar as informações extras.

Mesmo em um domínio localmente estruturado, só obteremos uma rede bayesiana se ordenarmos bem para escolher o nó. O que acontecerá se escolhermos a ordem errada? Vamos considerar novamente o exemplo do alarme contra roubo. Suponha que decidimos adicionar os nós na ordem *MariaLiga*, *JoãoLiga*, *Alarme*, *Roubo*, *Terremoto*. Nesse caso, obtemos a rede um pouco mais complicada mostrada na Figura 14.3(a). O processo se desenvolve assim:

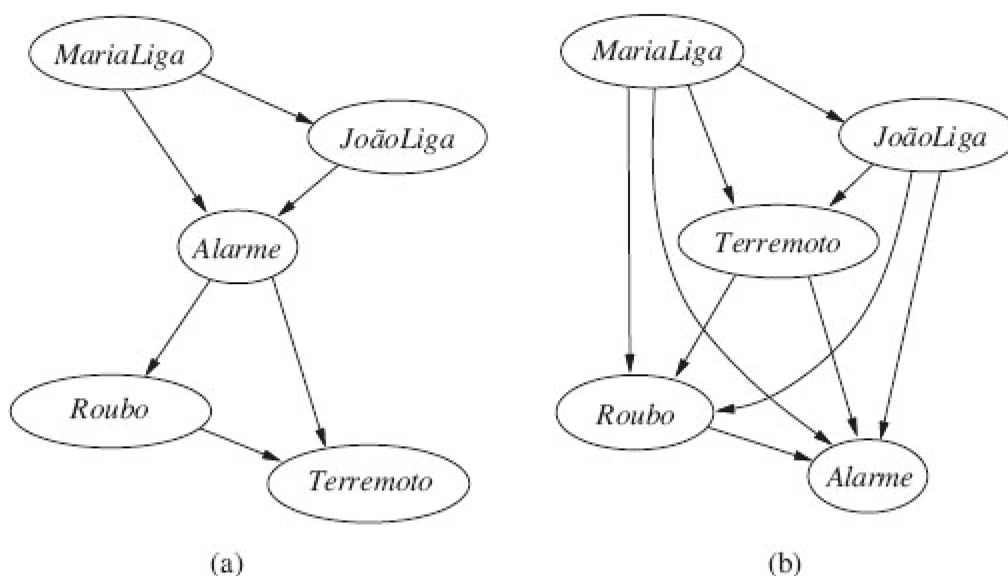


Figura 14.3 A estrutura de rede depende da ordem de introdução. Em cada rede, introduzimos nós na ordem de cima para baixo.

- Adicionando *MariaLiga*: não há pais.
- Adicionando *JoãoLiga*: se Maria liga, isso provavelmente significa que o alarme soou, e é claro que tornaria mais provável a ligação de João. Portanto, *JoãoLiga* precisa de *MariaLiga* como pai.
- Adicionando *Alarme*: é claro que, se ambos ligarem, será mais provável que o alarme tenha soado do que se apenas um ou nenhum deles ligar; assim, precisamos de *MariaLiga* e *JoãoLiga* como pais.
- Adicionando *Roubo*: se tivéssemos conhecimento do estado do alarme, a ligação de João ou de Maria poderia nos dar informações sobre o ruído da campainha de nosso telefone ou sobre a

música de Maria, mas não sobre roubo:

$$\mathbf{P}(Roubo \mid Alarme, JoãoLiga, MariaLiga) = \mathbf{P}(Roubo \mid Alarme).$$

Consequentemente precisamos apenas de *Alarme* como pai.

- Adicionando *Terremoto*: se o alarme estiver ligado, é mais provável que tenha havido um terremoto. (O alarme é uma espécie de detector de terremotos.) Porém, se soubermos que houve um roubo, isso explica o alarme, e a probabilidade de um terremoto estaria apenas ligeiramente acima da normal. Por conseguinte, precisamos de *Alarme* e *Roubo* como pais.

A rede resultante terá dois outros vínculos além da rede original da Figura 14.2 e exigirá outras três probabilidades para ser especificada. Pior ainda, alguns dos vínculos representam relacionamentos tênues que exigem julgamentos de probabilidade difíceis e antinaturais, como a avaliação da probabilidade de *Terremoto*, dados *Roubo* e *Alarme*. Esse fenômeno é bastante geral e está relacionado à distinção entre modelos causais e modelos de diagnóstico introduzidos na Seção 13.5.1 (veja também o Exercício 8.13). Se tentarmos construir um modelo de diagnóstico com vínculos de sintomas para causas (como, por exemplo, de *MariaLiga* para *Alarme* ou de *Alarme* para *Roubo*), acabaremos sendo obrigados a especificar dependências adicionais entre causas que de outra forma seriam independentes (e, com frequência, também entre sintomas que ocorrem separadamente). *Se nos fixarmos em um modelo causal, acabaremos tendo de especificar uma quantidade menor de números, e os números frequentemente serão mais fáceis de apresentar*. Por exemplo, no domínio da medicina, foi demonstrado por Tversky e Kahneman (1982) que os médicos especialistas preferem apresentar julgamentos de probabilidade para regras causais, em vez de fazê-lo para regras de diagnóstico.

A Figura 14.3(b) mostra uma ordenação de nós muito ruim: *MariaLiga*, *JoãoLiga*, *Terremoto*, *Roubo*, *Alarme*. Essa rede exige que sejam especificadas 31 probabilidades distintas — exatamente a mesma quantidade que a da distribuição conjunta total. No entanto, é importante perceber que qualquer das três redes pode representar *exatamente a mesma distribuição conjunta*. As duas últimas versões simplesmente deixam de representar todos os relacionamentos de independência condicional e, consequentemente, acabam por especificar em vez disso muitos números desnecessários.

14.2.2 Relações de independência condicional em redes bayesianas

Fornecemos uma semântica “numérica” para redes bayesianas em termos da representação da distribuição conjunta total, como na Equação 14.2. Usando essa semântica para derivar um método com a finalidade de construir redes bayesianas, fomos levados à consequência de que um nó é condicionalmente independente de seus predecessores, dados seus pais. Ocorre que também podemos seguir o sentido inverso. Podemos começar de uma semântica “topológica” que especifique os relacionamentos de independência condicional codificados pela estrutura de grafo e, a partir deles, podemos derivar a semântica “numérica”. A semântica topológica[2] especifica que cada variável é condicionalmente independente de seus não **descendentes**, dados seus pais. Por exemplo,

na Figura 14.2, *JoãoLiga* é independente de *Roubo*, *Terremoto* e *MariaLiga* dado o valor de *Alarme*. A definição está ilustrada na Figura 14.4(a). Dessas afirmações de independência condicional e da interpretação dos parâmetros da rede $\theta(X_i \mid Pais\ (X_i))$ como especificações das probabilidades condicionais $\mathbf{P}(X_i \mid Pais\ (X_i))$, a distribuição conjunta completa dada na Equação 14.2 pode ser reconstruída. Nesse sentido, as semânticas "numérica" e a "topológica" são equivalentes.

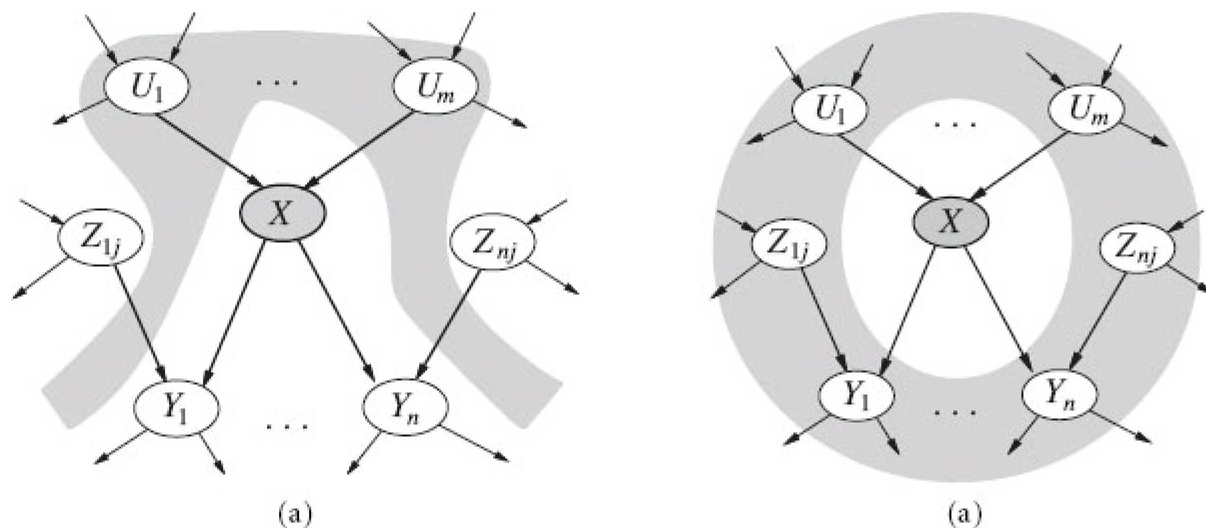


Figura 14.4 (a) Um nó X é condicionalmente independente de seus não descendentes (por exemplo, os nós Z_{ij}) dados seus pais (os nós U_i mostrados na área cinza). (b) Um nó X é condicionalmente independente de todos os outros nós da rede, dada sua cobertura de Markov (a área cinza).

A semântica topológica implica outra propriedade importante de independência: um nó é condicionalmente independente de todos os outros nós na rede, dados seus pais, filhos e pais dos filhos, isto é, dada a **cobertura de Markov** (o Exercício 14.7 pede para provar isso). Por exemplo, *Roubo* é independente de *JoãoLiga e MariaLiga*, dados *Alarme e Terremoto*. Essa propriedade está ilustrada na Figura 14.4(b).

14.3 REPRESENTAÇÃO EFICIENTE DE DISTRIBUIÇÕES CONDICIONAIS

Ainda que o número máximo de pais k seja reduzido, o preenchimento da TPC para um nó exige até $O(2^k)$ números e talvez muita experiência com todos os casos de condicionamento possíveis. De fato, esse é um cenário de pior caso, em que o relacionamento entre os pais e o filho é completamente arbitrário. Em geral, tais relacionamentos podem ser descritos por uma **distribuição canônica** que se ajusta a alguma forma-padrão. Em tais casos, a tabela completa pode ser especificada definindo-se o padrão e talvez fornecendo-se alguns parâmetros — o que é muito mais fácil que fornecer um número exponencial de parâmetros.

O exemplo mais simples é fornecido por **nós determinísticos**. Um nó determinístico tem seu valor especificado exatamente pelos valores de seus pais, sem qualquer incerteza. O relacionamento pode ser lógico: por exemplo, o relacionamento entre os nós pais *Canadense*, *Americano*, *Mexicano* e o nó filho *Norte-americano* é simplesmente o fato de que o filho é a disjunção dos pais. O relacionamento também pode ser numérico: por exemplo, se os nós pais são os preços de um modelo

específico de automóvel em diversos revendedores e o nó filho é o preço que um caçador de ofertas acaba pagando, o nó filho é o valor mínimo entre os valores pais; ou, então, se os nós pais são os afluentes — ou fluxos de entrada (rios, córregos, precipitações) de um lago — e os escoadouros — ou fluxos de saída (rios, evaporação, vazamentos) do lago — e o filho é a mudança no nível de água do lago, então o valor do filho é a soma dos fluxos de entrada menos a soma dos fluxos de saída.

Os relacionamentos incertos frequentemente podem ser caracterizados pelos chamados relacionamentos lógicos "ruidosos". O exemplo-padrão é a relação **OU ruidoso**, uma generalização do OU lógico. Em lógica proposicional, poderíamos dizer que *Febre* é verdadeira se e somente se *Resfriado*, *Gripe* ou *Malária* é verdadeira. O modelo OU ruidoso permite a incerteza sobre a habilidade de cada pai para fazer o filho ser verdadeiro — o relacionamento causal entre pai e filho pode ser *inibido*, e assim um paciente pode ter um resfriado, mas não apresentar febre. O modelo faz duas suposições. Primeiro, ele pressupõe que todas as causas possíveis estão listadas (se algo estiver faltando, sempre podemos adicionar um chamado **nó de vazamento** que cobre "causas diversas"). Em segundo lugar, ele pressupõe que a inibição de cada pai é independente da inibição de quaisquer outros pais: por exemplo, o que inibe *Malária* de causar uma febre é independente do que inibe *Gripe* de causar uma febre. Dadas essas suposições, *Febre* é *falsa* se e somente se todos os seus pais *verdadeiros* são inibidos, e a probabilidade de ocorrer isso é o produto das probabilidades de inibição θ para cada pai. Vamos supor que essas probabilidades de inibição individuais sejam:

$$q_{\text{resfriado}} = P(\neg febre \mid resfriado, \neg gripe, \neg malária) = 0{,}6,$$
$$q_{\text{gripe}} = P(\neg febre \mid \neg resfriado, gripe, \neg malária) = 0{,}2,$$
$$q_{\text{malária}} = P(\neg febre \mid \neg resfriado, \neg gripe, malária) = 0{,}1.$$

Assim, a partir dessas informações e das suposições de OU ruidoso, é possível construir a TPC inteira. A regra geral é que:

$$P(x_i \mid pais(X_i)) = 1 - \prod_{\{j : X_j = \text{ verdadeiro}\}} q_j \,,$$

onde o produto é obtido dos pais que são definidos como verdadeiro para essa linha da TPC. A tabela a seguir ilustra esse cálculo:

Resfriado	*Gripe*	*Malária*	*P*(*Febre*)	*P*¬ (*Febre*)
F	F	F	0,0	1,0
F	F	V	0,9	**0,1**
F	V	F	0,8	**0,2**
F	V	V	0,98	0,02 = 0,2 × 0,1
V	F	F	0,4	**0,6**
V	F	V	0,94	0,06 = 0,6 × 0,1
V	V	F	0,88	0,12 = 0,6 × 0,2
V	V	V	0,988	0,012 = 0,6 × 0,2 × 0,1

Em geral, relacionamentos lógicos ruidosos em que uma variável depende de *k* pais podem ser

descritos com o uso de $O(k)$ parâmetros, em vez de $O(2^k)$ para a tabela de probabilidade condicional completa. Isso torna muito mais fácil a avaliação e o aprendizado. Por exemplo, a rede CPCS (Pradhan *et al.*, 1994) utiliza distribuições de OU ruidoso e MAX ruidoso para modelar relacionamentos entre doenças e sintomas em medicina interna. Com 448 nós e 906 vínculos, ela exige apenas 8.254 valores, em vez de 133.931.430 para uma rede com TPCs completas.

Redes bayesianas com variáveis contínuas

Muitos problemas reais envolvem quantidades contínuas, como altura, massa, temperatura e dinheiro; de fato, grande parte da estatística lida com variáveis aleatórias cujos domínios são contínuos. Por definição, variáveis contínuas têm um número infinito de valores possíveis e, assim, é impossível especificar explicitamente probabilidades condicionais para cada valor. Um modo possível de manipular variáveis contínuas é evitá-las usando a **discretização**, isto é, repartindo os valores possíveis em um conjunto fixo de intervalos. Por exemplo, as temperaturas poderiam ser divididas dentre (<0°C), (0°C–100°C) e (>100°C). A discretização às vezes é uma solução adequada, mas frequentemente resulta em perda considerável de precisão e em TPCs muito grandes. A solução mais comum é definir famílias-padrão de funções de densidade de probabilidade (veja o Apêndice A) que são especificadas por um número finito de **parâmetros**. Por exemplo, uma distribuição gaussiana (ou normal) $N(\mu, \sigma^2)(x)$ tem a média μ e a variância σ^2 como parâmetros. Ainda uma outra solução — às vezes chamada representação **não paramétrica** — é definir implicitamente a distribuição condicional com uma coleção de instâncias, cada uma contendo os valores específicos das variáveis do pai e do filho. Essa abordagem será explorada no Capítulo 18.

Uma rede com variáveis discretas e contínuas é chamada **rede bayesiana híbrida**. Para especificar uma rede híbrida, temos de especificar dois novos tipos de distribuições: a distribuição condicional para uma variável contínua dados pais discretos ou contínuos e a distribuição condicional para uma variável discreta dados pais contínuos. Considere o exemplo simples da Figura 14.5, em que um cliente compra alguma fruta dependendo de seu custo que, por sua vez, depende do volume da colheita e do fato de o esquema de subsídios do governo estar em vigor. A variável *Custo* é contínua e tem pais contínuos e discretos; a variável *Compra* é discreta e tem pai contínuo.

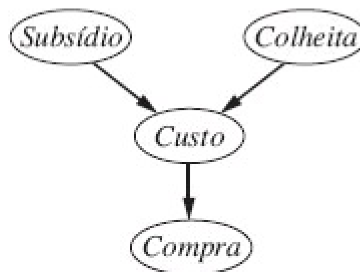


Figura 14.5 Uma rede simples com variáveis discretas (*Subsídio* e *Compra*) e variáveis contínuas (*Colheita* e *Custo*).

Para a variável *Custo*, precisamos especificar **P**(*Custo* | *Colheita*, *Subsídio*). O pai discreto é manipulado por enumeração explícita, ou seja, pela especificação de *P*(*Custo* | *Colheita*, *subsídio*) e de *P*(*Custo* | *Colheita*, ¬*subsídio*). Para tratar *Colheita*, especificamos como a distribuição sobre o custo *c* depende do valor contínuo *h* de *Colheita*. Em outras palavras, especificamos os *parâmetros* da distribuição de custo como uma função de *h*. A escolha mais comum é a **distribuição gaussiana**

linear, na qual o filho tem uma distribuição gaussiana cuja média μ varia linearmente com o valor do pai e cujo desvio-padrão σ é fixo. Precisamos de duas distribuições, uma para *subsídio* e uma para ¬*subsídio*, com parâmetros diferentes:

$$P(c \mid h, subsídio) = N(a_t h + b_t, \sigma_t^2)(c) = \frac{1}{\sigma_t\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_t h+b_t)}{\sigma_t}\right)^2}$$

$$P(c \mid h, \neg subsídio) = N(a_f h + b_f, \sigma_f^2)(c) = \frac{1}{\sigma_f\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_f h+b_f)}{\sigma_f}\right)^2} .$$

Para esse exemplo, a distribuição condicional para *Custo* é especificada pela nomenclatura da distribuição gaussiana linear, fornecendo-se os parâmetros a_t, b_t, Σ_t, a_f, b_f e Σ_f. As Figuras 14.6(a) e (b) mostram esses dois relacionamentos. Note que, em cada caso, a inclinação é negativa porque o preço diminui à medida que a quantidade fornecida aumenta. (É claro que a suposição de linearidade implica que o preço se torna negativo em algum momento; o modelo linear só é razoável se o volume da colheita for limitado a um intervalo estreito.) A Figura 14.6(c) mostra a distribuição $P(c \mid h)$, calculada pela média sobre os dois valores possíveis de *Subsídio* e supondo que cada um deles tenha probabilidade *a priori* igual a 0,5. Isso mostra que até mesmo com modelos muito simples podem ser representadas distribuições bastante interessantes.

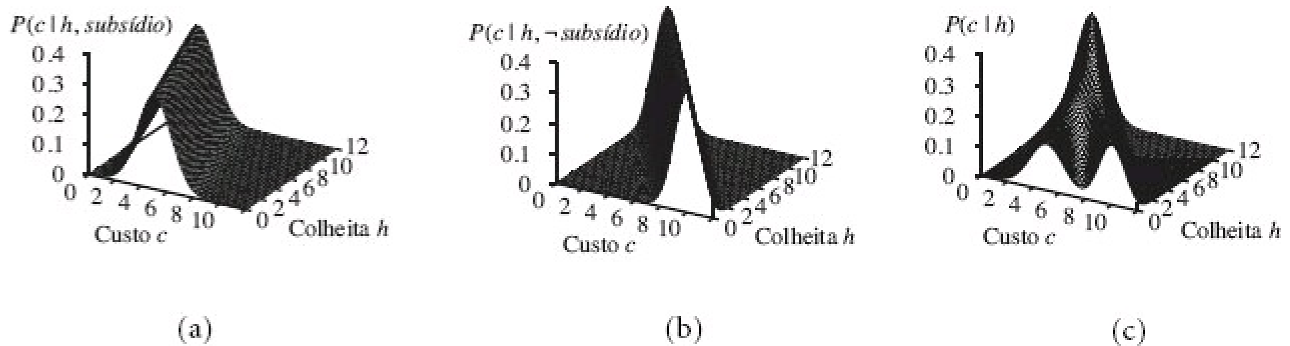


Figura 14.6 Os grafos em (a) e (b) mostram a distribuição de probabilidade sobre *Custo* como uma função do volume da *Colheita*, com *Subsídio* verdadeiro e falso, respectivamente. O grafo (c) mostra a distribuição *P*(*Custo* | *Colheita*), obtida pelo somatório sobre os dois casos de subsídios.

A distribuição gaussiana condicional linear tem algumas propriedades especiais. Uma rede que contém apenas variáveis contínuas com distribuições gaussianas lineares tem uma distribuição conjunta que é uma distribuição gaussiana multivariada (veja o Apêndice A) sobre todas as variáveis (Exercício 14.9). Além disso, dada alguma evidência, a distribuição posterior também tem essa propriedade.[3] Quando são adicionadas variáveis discretas como pais (não como filhos) de variáveis contínuas, a rede define uma **distribuição gaussiana condicional**, ou GC: dada qualquer atribuição às variáveis discretas, a distribuição sobre as variáveis contínuas é uma distribuição gaussiana multivariada.

Agora, vamos estudar as distribuições para variáveis discretas com pais contínuos. Por exemplo, considere o nó *Compras* da Figura 14.5. Parece razoável supor que o cliente comprará se o custo for baixo e não comprará se ele for alto, e que a probabilidade de compra varia suavemente em alguma região intermediária. Em outras palavras, a distribuição condicional é semelhante a uma função de limiar "suave". Um modo de criar limiares suaves é usar a *integral* da distribuição normal padrão:

$$\Phi(x) = \int_{-\infty}^{x} N(0,1)(x)dx \; .$$

Então, a probabilidade de *Compras* dado *Custo* poderia ser:

$$P(compras \mid Custo = c) = \Phi((-c + \mu)/\Sigma),$$

o que significa que o limiar de custo ocorre em torno de μ, que a largura da região de limiar é proporcional a Σ e que a probabilidade de compra diminui à medida que o custo aumenta.

Essa **distribuição probit (**abreviatura de "unidade de probabilidade") está ilustrada na Figura 14.7(a). A forma pode ser justificada pela proposição de que o processo de decisão subjacente tem um limiar difícil, mas que a posição precisa do limiar está sujeita a ruído gaussiano aleatório.

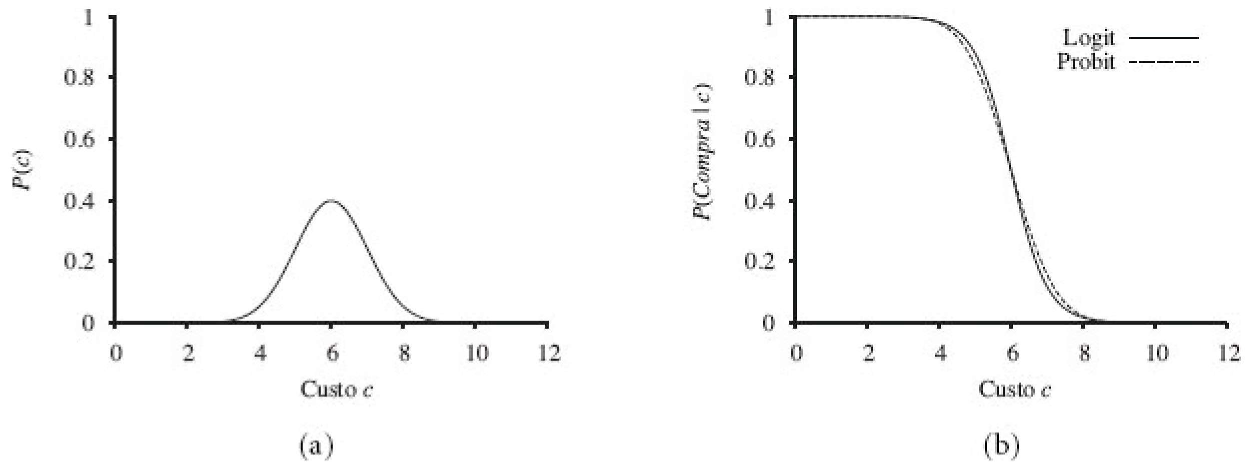


Figura 14.7 (a) Uma distribuição (gaussiana) normal para o limite de custo, centrada em $\mu = 6{,}0$, com desvio-padrão $\Sigma = 1{,}0$. (b) Distribuições Logit e Probit para a probabilidade de *compras* dado *custo*, para os parâmetros $\mu = 6{,}0$ e $\Sigma = 1{,}0$.

Uma alternativa para o modelo probit é a **distribuição logit**, que utiliza a função logística $1/(1 + e^{-x})$ para produzir um limiar suave:

$$P(\,compras \mid Custo = c) = \frac{1}{1 + exp(-2\frac{-c+\mu}{\sigma})} \; .$$

Isso está ilustrado na Figura 14.7(b). As duas distribuições parecem semelhantes, mas, na realidade, a distribuição logit tem extremidades muito mais longas. Com frequência, a distribuição probit se ajusta melhor a situações reais, embora às vezes seja mais fácil lidar matematicamente com a distribuição logit. Ela é amplamente utilizada em redes neurais (Capítulo 20). Tanto probit quanto logit podem ser generalizadas para manipular vários pais contínuos, tomando-se uma combinação linear dos valores dos pais.

14.4 INFERÊNCIA EXATA EM REDES BAYESIANAS

A tarefa básica para qualquer sistema de inferência probabilístico é calcular a distribuição de probabilidade posterior para um conjunto de **variáveis de consulta**, dado algum **evento** observado,

isto é, alguma atribuição de valores a um conjunto de **variáveis de evidência**. Para simplificar a apresentação, consideraremos apenas uma variável de consulta por vez; os algoritmos podem ser facilmente estendidos para consultas com variáveis múltiplas. Utilizaremos a notação introduzida no Capítulo 13: X denota a variável de consulta; **E** denota o conjunto de variáveis de evidência $E_1, \ldots, E_m$ e **e** é um evento específico observado; **Y** denotará as variáveis que não são de evidência $Y_1, \ldots, Y_l$ (às vezes chamadas **variáveis ocultas**). Desse modo, o conjunto completo de variáveis $\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$. Uma consulta típica busca a distribuição de probabilidade posterior $\mathbf{P}(X \mid \mathbf{e})$.

Na rede de alarme contra roubo, poderíamos observar o evento em que *JoãoLiga* = *verdadeiro* e *MariaLiga* = *verdadeiro*. Então, poderíamos buscar, digamos, a probabilidade de ter ocorrido um roubo:

$$\mathbf{P}(Roubo \mid JoãoLiga = verdadeiro, MariaLiga = verdadeiro) = \langle 0{,}284, 0{,}716 \rangle.$$

Nesta seção, discutiremos algoritmos exatos para calcular probabilidades posteriores e consideraremos a complexidade dessa tarefa. Ocorre que o caso geral é intratável e, assim, a Seção 14.5 estuda métodos para inferência aproximada.

14.4.1 Inferência por enumeração

O Capítulo 13 explicou que qualquer probabilidade condicional pode ser calculada pelo somatório de termos da distribuição conjunta total. Mais especificamente, uma consulta $\mathbf{P}(X \mid \mathbf{e})$ pode ser respondida com a utilização da Equação 13.9, que repetimos aqui por conveniência:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha\, \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}) .$$

Agora, uma rede bayesiana fornece uma representação completa da distribuição conjunta total. Mais especificamente, a Equação 14.2 mostra que os termos $P(x, \mathbf{e}, \mathbf{y})$ na distribuição conjunta podem ser escritos como produtos de probabilidades condicionais da rede. Por conseguinte, *uma consulta pode ser respondida com o uso de uma rede bayesiana, calculando-se somas de produtos de probabilidades condicionais da rede*.

Considere a consulta $\mathbf{P}(Roubo \mid JoãoLiga = verdadeiro, MariaLiga = verdadeiro)$. As variáveis ocultas para essa consulta são *Terremoto* e *Alarme*. Da Equação 13.9, usando letras iniciais para representar as variáveis com o objetivo de encurtar as expressões, temos:[4]

$$\mathbf{P}(B \mid j, m) = \alpha\, \mathbf{P}(B, j, m) = \alpha \sum_{e} \sum_{a} \mathbf{P}(B, j, m, e, a,) .$$

A semântica das redes bayesianas (Equação 14.2) nos dá então uma expressão em termos de entradas de TPC. Por simplicidade, faremos isso apenas para *Roubo* = *verdadeiro*:

$$P(b \mid j, m) = \alpha \sum_{e} \sum_{a} P(b) P(e) P(a \mid b, e) P(j \mid a) P(m \mid a) .$$

Para calcular essa expressão, temos de somar quatro termos, cada um calculado pela multiplicação de cinco números. No pior caso, em que teremos de efetuar o somatório de quase todas as variáveis, a complexidade do algoritmo para uma rede com n variáveis booleanas será $O(n2^n)$.

Uma melhoria pode ser obtida a partir das observações simples a seguir: o termo $P(b)$ é uma constante e pode ser movido para fora dos somatórios sobre a e e, e o termo $P(e)$ pode ser movido para fora do somatório sobre a. Consequentemente, temos:

$$P(b \mid j, m) = \alpha\, P(b) \sum_{e} P(e) \sum_{a} P(a \mid b, e) P(j \mid a) P(m \mid a)\,. \qquad (14.4)$$

Essa expressão pode ser avaliada por meio de um laço repetitivo através das variáveis em ordem, multiplicando entradas de TPC à medida que avançarmos. Para cada somatório, também precisamos executar um laço sobre os valores possíveis da variável. A estrutura dessa computação é mostrada na Figura 14.8. Usando os números da Figura 14.2, obtemos $P(b \mid j, m) = \alpha \times 0{,}00059224$. A computação correspondente para $\neg b$ produz $\alpha \times 0{,}0014919$; por conseguinte,

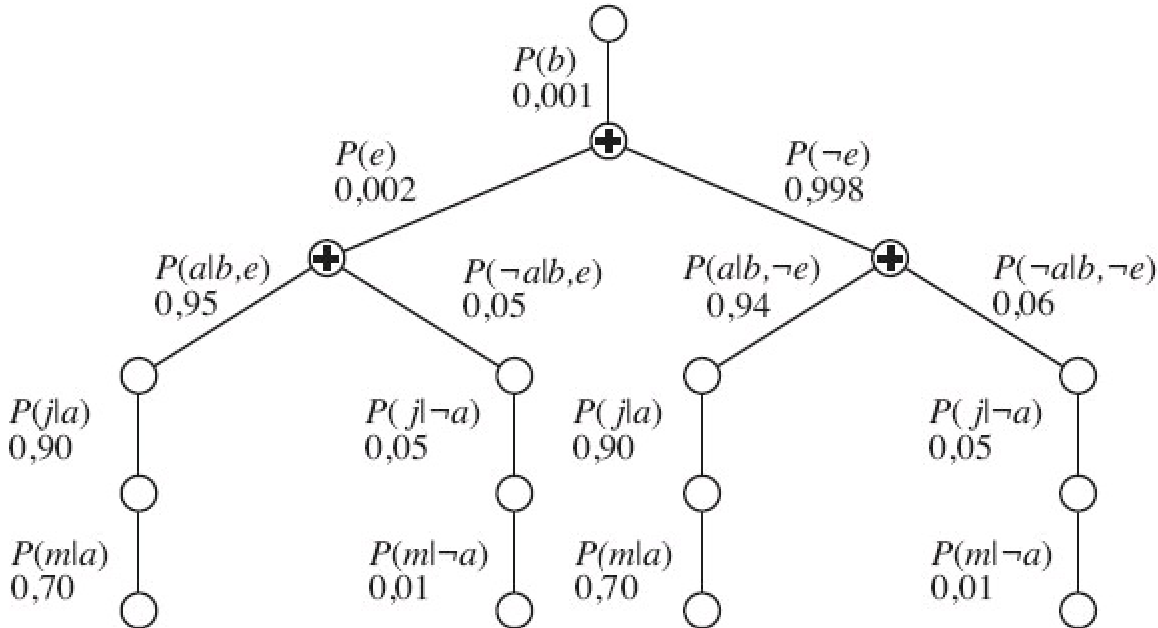


Figura 14.8 Estrutura da expressão mostrada na Equação 14.4. A avaliação prossegue de cima para baixo, multiplicando valores ao longo de cada caminho e efetuando o somatório nos nós identificados com "+". Note a repetição dos caminhos para j e m.

$$\mathbf{P}(B \mid j, m) = \alpha \langle 0{,}00059224,\ 0{,}0014919 \rangle \approx \langle 0{,}284,\ 0{,}716 \rangle.$$

Ou seja, a chance de um roubo, dadas as ligações de ambos os vizinhos, é de aproximadamente 28%.

O processo de avaliação para a expressão da Equação 14.4 é mostrado como uma árvore de expressões na Figura 14.8. O algoritmo ASK-ENUMERAÇÃO da Figura 14.9 avalia tais árvores usando a recursão primeiro na profundidade. O algoritmo é muito semelhante em estrutura ao algoritmo de retrocesso para a resolução de PSRs (Figura 6.5) e ao algoritmo de satisfatibilidade DPLL (Figura 7.17).

função ASK-ENUMERAÇÃO(X, **e**, rb) **retorna** uma distribuição sobre X

entradas: X, a variável de consulta

e, valores observados para variáveis **E**

rb, uma rede bayesiana com variáveis $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$ /* **Y** = *variáveis ocultas* */

$\mathbf{Q}(X) \leftarrow$ uma distribuição sobre X, inicialmente vazia

para cada valor x_i de X **faça**

estender **e** com valor *xi* para X

$\mathbf{Q}(xi) \leftarrow$ ENUMERAR-TODOS(*bn.*VARS, e_{xi})

Where $\mathbf{e}_{xi}$ é **e** estendido com $X = x_i$

retornar NORMALIZAR($\mathbf{Q}(X)$)

---

função ENUMERAR-TODOS(*vars*, **e**) **retorna** um número real

se VAZIO?(*vars*) **então retornar** 1,0

$Y \leftarrow$ PRIMEIRO(*vars*)

se Y tem valor y em **e**

então retornar $P(y \mid pais(Y)) \times$ ENUMERAR-TODOS(RESTO(*vars*), **e**)

senão retornar Σ_y P(y | pais(y)) × ENUMERAR-TODOS (vars), e_y)

onde $\mathbf{e}_{yi}$ é **e** estendido com $Y = y_i$

Figura 14.9 O algoritmo de enumeração para responder a consultas sobre redes bayesianas.

Desse modo, a complexidade de espaço de ASK-ENUMERAÇÃO só é linear no número de variáveis: efetivamente, o algoritmo efetua o somatório sobre a distribuição conjunta total sem jamais construí-la de forma explícita. Infelizmente, sua complexidade de tempo para uma rede com n variáveis booleanas é sempre $O(2^n)$ — melhor que o valor $O(n2^n)$ para a abordagem simples descrita anteriormente, mas ainda terrível.

Observe que a árvore da Figura 14.8 torna explícitas as *subexpressões repetidas* que são avaliadas pelo algoritmo. Os produtos $P(j \mid a)P(m \mid a)$ e $P(j \mid \neg a)P(m \mid \neg a)$ são calculados duas vezes, uma para cada valor de e. A próxima seção descreve um método geral que evita esse desperdício de computações.

14.4.2 O algoritmo de eliminação de variáveis

O algoritmo de enumeração pode ser substancialmente melhorado eliminando-se cálculos repetidos do tipo ilustrado na Figura 14.8. A ideia é simples: efetuar o cálculo apenas uma vez e guardar os resultados para uso posterior. Essa é uma forma de programação dinâmica. Existem várias versões dessa abordagem; apresentamos o algoritmo de **eliminação de variáveis**, que é a mais simples. A eliminação de variáveis funciona avaliando expressões como a Equação 14.4 na ordem *da direita para a esquerda* (isto é, *de baixo para cima* na Figura 14.8). Os resultados intermediários são armazenados, e os somatórios sobre cada variável são efetuados apenas para as porções da

expressão que dependem da variável.

Vamos ilustrar esse processo para a rede de alarme contra roubo. Avaliamos a expressão:

$$\mathbf{P}(B \mid j, m) = \alpha \underbrace{\mathbf{P}(B)}_{\mathbf{f}_1(B)} \sum_e \underbrace{P(e)}_{\mathbf{f}_2(E)} \sum_a \underbrace{\mathbf{P}(a \mid B, e)}_{\mathbf{f}_3(A,B,E)} \underbrace{P(j \mid a)}_{\mathbf{f}_4(A)} \underbrace{P(m \mid a)}_{\mathbf{f}_5(A)} .$$

Observe que identificamos cada parte da expressão com o nome do **fator** correspondente; cada fator é uma matriz indexada pelos valores das variáveis de seu argumento. Por exemplo, os fatores f_4(A) e f_5(A) correspondentes a $P(j \mid a)$ e $P(m \mid a)$ dependem apenas de *A* porque *J* e *M* são fixados pela consulta. Eles são, portanto, vetores de dois elementos:

$$\mathbf{f}_4(A) = \begin{pmatrix} P(j \mid a) \\ P(j \mid \neg a) \end{pmatrix} = \begin{pmatrix} 0{,}90 \\ 0{,}05 \end{pmatrix} \qquad \mathbf{f}_5(A) = \begin{pmatrix} P(m \mid a) \\ P(m \mid \neg a) \end{pmatrix} = \begin{pmatrix} 0{,}70 \\ 0{,}01 \end{pmatrix} .$$

$\mathbf{f}_3(A, B, E)$ será uma matriz 2 × 2 × 2, que é difícil de mostrar na página impressa. (O "primeiro" elemento é dado por $P(a \mid b, e) = 0{,}95$ e o "último" por $P(\neg a \mid \neg b, \neg e) = 0{,}999$.) Em termos de fatores, a expressão de consulta é escrita como

$$\mathbf{P}(B \mid j, m) = \alpha \, \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \sum_a \mathbf{f}_3(A, B, E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A)$$

onde o operador "×" não é uma matriz ordinária de multiplicação, mas a operação do **produto pontual**, que será descrito brevemente.

O processo de avaliação é um processo de somar variáveis (da direita para a esquerda) dos produtos de fatores pontuais para produzir fatores novos, eventualmente gerando um fator que seja uma solução, ou seja, a distribuição posterior sobre a variável de consulta. As etapas são as seguintes:

- Em primeiro lugar, somamos A do produto de f_3, f_4 e f_5. Isso nos dá um fator novo 2 × 2 f_6 (*B*, *E*) cujas faixas de índices vão de *B* a *E*:

$$\begin{aligned} \mathbf{f}_6(B, E) &= \sum_a \mathbf{f}_3(A, B, E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A) \\ &= (\mathbf{f}_3(a, B, E) \times \mathbf{f}_4(a) \times \mathbf{f}_5(a)) + (\mathbf{f}_3(\neg a, B, E) \times \mathbf{f}_4(\neg a) \times \mathbf{f}_5(\neg a)) . \end{aligned}$$

  Agora ficamos com a expressão

$$\mathbf{P}(B \mid j, m) = \alpha \, \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B, E) .$$

- Em seguida, somamos E do produto de $\mathbf{f}_2$ e $\mathbf{f}_6$:

$$\begin{aligned} \mathbf{f}_7(B) &= \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B, E) \\ &= \mathbf{f}_2(e) \times \mathbf{f}_6(B, e) + \mathbf{f}_2(\neg e) \times \mathbf{f}_6(B, \neg e) . \end{aligned}$$

Fica a expressão

$$\mathbf{P}(B \mid j, m) = \alpha\, \mathbf{f}_1(B) \times \mathbf{f}_7(B)$$

que pode ser avaliada extraindo o produto pontual e normalizando o resultado.

Examinando essa sequência de etapas, vemos que existem duas operações computacionais básicas exigidas: o produto pontual de um par de fatores e o somatório de uma variável de um produto de fatores. A próxima seção descreve cada uma dessas operações.

Operações com fatores

O produto pontual de dois fatores $\mathbf{f}_1$ e $\mathbf{f}_2$ gera um novo fator **f** cujas variáveis são a *união* das variáveis contidas em $\mathbf{f}_1$ e $\mathbf{f}_2$ e cujos elementos são dados pelo produto dos elementos correspondentes em dois fatores. Suponha que os dois fatores tenham variáveis $Y_1, \ldots, Y_k$ em comum. Então, temos:

$$\mathbf{f}(X_1 \ldots X_j, Y_1 \ldots Y_k, Z_1 \ldots Z_l) = \mathbf{f}_1(X_1 \ldots X_j, Y_1 \ldots Y_k)\, \mathbf{f}_2(Y_1 \ldots Y_k, Z_, \ldots Z_l).$$

Se todas as variáveis forem binárias, então $\mathbf{f}_1$ e $\mathbf{f}_2$ terão 2^{j+k} e 2^{k+l} entradas, respectivamente, e o produto pontual terá 2^{j+k+l} entradas. Por exemplo, dados dois fatores $\mathbf{f}_1(A, B)$ e $\mathbf{f}_2(B, C)$, o produto pontual $f_1 \times f_2 = f_3\,(A, B, C)$ tem $2^{1+1+1} = 8$ entradas, como ilustrado na Figura 14.10. Note que o fator resultante de um produto pontual pode conter mais variáveis que qualquer um dos fatores que estão sendo multiplicadas e que o tamanho de um fator é exponencial ao número de variáveis. Esse é o lugar onde a complexidade de espaço e de tempo surge na variável algoritmo de eliminação.

A	*B*	$\mathbf{f}_1(A, B)$	*B*	*C*	$\mathbf{f}_2(B, C)$	*A*	*B*	*C*	$\mathbf{f}_3(A, B, C)$
						V	V	V	0,3 × 0,2 = 0,06
						V	V	F	0,3 × 0,8 = 0,24
V	V	0,3	V	V	0,2	V	F	V	0,7 × 0,6 = 0,42
V	F	0,7	V	F	0,8	V	F	F	0,7 × 0,4 = 0,28
F	V	0,9	F	V	0,6	F	V	V	0,9 × 0,2 = 0,18
F	F	0,1	F	F	0,4	F	V	F	0,9 × 0,8 = 0,72
						F	F	V	0,1 × 0,6 = 0,06
						F	F	F	0,1 × 0,4 = 0,04

Figure 14.10 Ilustração da multiplicação pontual: $\mathbf{f}_1(A,B) \times \mathbf{f}_2(B,C) = \mathbf{f}_3(A,B,C)$.

A soma de uma variável de um produto de fatores é efetuada pela adição de submatrizes que são formadas fixando a variável em cada um de seus valores por vez. Por exemplo, para a soma de *A* $\mathbf{f}_3(A, B, C)$, escrevemos

$$\begin{aligned}\mathbf{f}(B,C) &= \sum_a \mathbf{f}_3(A,B,C) = \mathbf{f}_3(a,B,C) + \mathbf{f}_3(\neg a,B,C) \\ &= \begin{pmatrix}0{,}06 & 0{,}24\\0{,}42 & 0{,}28\end{pmatrix} + \begin{pmatrix}0{,}18 & 0{,}72\\0{,}06 & 0{,}04\end{pmatrix} = \begin{pmatrix}0{,}24 & 0{,}96\\0{,}48 & 0{,}32\end{pmatrix}.\end{aligned}$$

O único truque é notar que qualquer fator que *não* dependa da variável a ser somada pode ser movido para fora do somatório. Por exemplo, se quiséssemos somar primeiro E na rede de roubo, a parte relevante da expressão seria

$$\sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A) = \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E)\,.$$

Agora o produto pontual dentro do somatório é calculado, e a variável é a soma da matriz resultante.

Note que as matrizes *não* são multiplicadas até precisarmos efetuar o somatório de uma variável a partir do produto acumulado. Nesse ponto, multiplicamos apenas as matrizes que incluem a variável a ser totalizada.

Dadas as rotinas para produto pontual e somatório, o próprio algoritmo de eliminação de variáveis pode ser escrito de forma bastante simples, como mostra a Figura 14.11.

função ASK-ELIMINAÇÃO(X, **e**, *rb*) **retorna** uma distribuição sobre X
  entradas: X, a variável de consulta
    e, variáveis observadas da variável **E**
    rb, uma rede bayesiana especificando a distribuição conjunta $\mathbf{P}(X_1, ..., X_n)$

  fatores ← [ ]
  para cada *var* **em** ORDEM(*rb*.VARS) **faça**
    fatores ← [CRIAR-FATOR (*var*, **e**) *fatores*]
    se *var* é uma variável oculta **então** *fatores* ← SOMAR(*var* | *fatores*)
  retornar NORMALIZAR(PRODUTO-PONTUAL(*fatores*))

Figura 14.11 Algoritmo de eliminação de variáveis para inferência nas redes bayesianas.

Ordenação e relevância de variáveis

O algoritmo na Figura 14.11 inclui uma função ORDEM não especificada para escolher uma ordenação para as variáveis. Cada escolha de ordenação produz um algoritmo válido, mas ordenações diferentes fazem com que seja gerado durante o cálculo fatores intermediários diferentes. Por exemplo, no cálculo mostrado anteriormente, eliminamos A antes de E; se fizermos o contrário, o cálculo torna-se

$$\mathbf{P}(B \mid j,m) = \alpha\, \mathbf{f}_1(B) \times \sum_a \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E)\,,$$

durante o qual um novo fator $\mathbf{f}_6(A, B)$ será gerado.

Em geral, os requisitos de tempo e de espaço de eliminação de variáveis são dominados pelo tamanho do maior fator construído durante a operação do algoritmo. Este, por sua vez, é determinado pela ordem de eliminação de variáveis e pela estrutura da rede. Determinar a ordem ótima é intratável, mas várias heurísticas boas ficam disponíveis. Um método bastante eficaz é o guloso: eliminar qualquer variável que minimize o tamanho do próximo fator a ser construído.

Vamos considerar mais uma consulta: **P**(*JoãoLiga* | *Roubo* = *verdadeira*). Como sempre, o primeiro passo é escrever o somatório aninhado:

$$\mathbf{P}(J \mid b) = \alpha\, P(b) \sum_{e} P(e) \sum_{a} P(a \mid b, e) \mathbf{P}(J \mid a) \sum_{m} P(m \mid a)\ .$$

Se avaliarmos essa expressão da direita para a esquerda, notaremos algo interessante: $\Sigma_m P(m|a)$ é igual a 1 por definição! Consequentemente, não havia nenhuma necessidade de incluí-lo; a variável M é *irrelevante* para essa consulta. Outro modo de dizer isso é afirmar que o resultado da consulta *P*(*JoãoLiga* | *Roubo* = *verdadeira*) ficará inalterado se removermos completamente da rede *MariaLiga*. Em geral, podemos remover qualquer nó de folha que não seja uma variável de consulta ou uma variável de evidência. Depois de sua remoção, pode haver mais alguns nós folhas, e esses também podem ser irrelevantes. Continuando com esse processo, descobriremos por fim que *toda variável que não é um ancestral de uma variável de consulta ou de uma variável de evidência é irrelevante para a consulta*. Um algoritmo de eliminação de variáveis pode portanto remover todas essas variáveis antes de avaliar a consulta.

14.4.3 A complexidade da inferência exata

A complexidade da inferência exata em redes bayesianas depende fortemente da estrutura da rede. A rede de alarme contra roubo da Figura 14.2 pertence à família de redes em que existe, no máximo, um caminho não orientado entre dois nós quaisquer na rede. Essas redes são chamadas **redes unicamente conexas** ou **poliárvores**, e têm uma propriedade particularmente interessante: *a complexidade de tempo e de espaço da inferência exata em poliárvores é linear em relação ao tamanho da rede*. Aqui, o tamanho é definido como o número de entradas de TPC; se o número de pais de cada nó estiver limitado por uma constante, a complexidade também será linear em relação ao número de nós.

No caso de **redes multiplamente conectadas**, como a da Figura 14.12(a), a eliminação de variáveis pode ter complexidade de tempo e de espaço exponencial no pior caso, mesmo quando o número de pais por nó é limitado. Isso não é surpreendente quando se considerar que, *pelo fato de incluir a inferência em lógica proposicional como um caso especial*, *a inferência em redes bayesianas é NP-difícil*. De fato, podemos mostrar (Exercício 14.16) que o problema é tão difícil quanto o de calcular o *número* de atribuições satisfatórias para uma fórmula de lógica proposicional. Isso significa que ele é #P-difícil ("número P difícil"), isto é, estritamente mais difícil que problemas NP-completos.

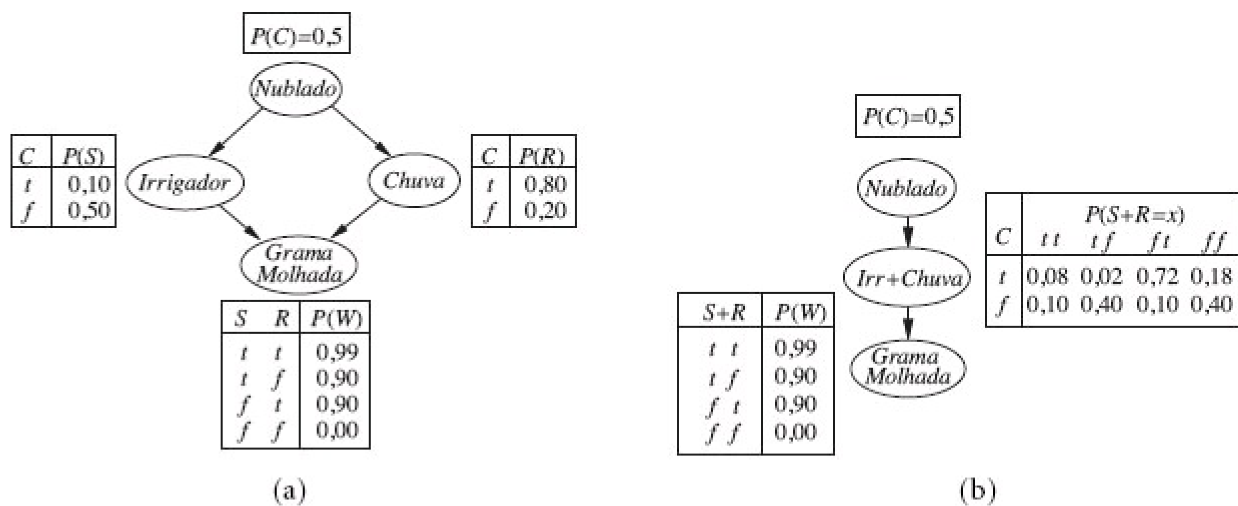


Figura 14.12 (a) Rede multiplamente conectada às tabelas de probabilidade condicional. (b) Formação de agrupamentos equivalentes a uma rede multiplamente conectada.

Existe uma conexão estreita entre a complexidade da inferência de redes bayesianas e a complexidade de problemas de satisfação de restrições (PSRs). Conforme discutimos no Capítulo 6, a dificuldade de resolução de um PSR discreto está relacionada ao quanto seu grafo de restrições é "semelhante a uma árvore". Medidas como **largura de rárvore**, que limitam a complexidade de resolução de um PSR, também podem ser aplicadas diretamente a redes bayesianas. Além disso, o algoritmo de eliminação de variáveis pode ser generalizado para resolver PSRs, bem como redes bayesianas.

14.4.4 Algoritmos de formação de agrupamentos

O algoritmo de eliminação de variáveis é simples e eficiente para responder a consultas individuais. Porém, se quisermos calcular as probabilidades posteriores para todas as variáveis em uma rede, talvez ele seja menos eficiente. Por exemplo, em uma rede de poliárvore, seria necessário emitir $O(n)$ consultas ao custo de $O(n)$ cada uma, totalizando um tempo igual a $O(n^2)$. Usando-se algoritmos de **formação de agrupamentos** (também conhecidos como algoritmos de **árvore de junção**), o tempo pode ser reduzido a $O(n)$. Por essa razão, esses algoritmos são amplamente usados em ferramentas comerciais de rede bayesiana.

A ideia básica da formação de agrupamentos é unir nós individuais da rede para formar nós de agrupamento, de tal modo que a rede resultante seja uma poliárvore. Por exemplo, a rede de várias conexões mostrada na Figura 14.12(a) pode ser convertida em uma poliárvore combinando-se os nós *Irrigador* e *Chuva* em um nó de agrupamento chamado *Irrigador*+*Chuva*, como mostra a Figura 14.12(b). Os dois nós booleanos são substituídos por um "meganó" que assume quatro valores possíveis: *tt*, *tf*, *ft* e *ff*. O meganó tem apenas um pai, a variável booleana *Nublado* e, assim, existem dois casos de condicionamento. Apesar do exemplo não mostrar isso, o processo de formação de agrupamentos sempre produz meganós que compartilham algumas variáveis.

Uma vez que a rede está em forma de poliárvore, é requerido um algoritmo de inferência de uso especial porque métodos de inferência ordinária não podem manusear meganós que compartilham variáveis uns com os outros. Em essência, o algoritmo é uma forma de propagação de restrições

(veja o Capítulo 6) em que as restrições garantem que os agrupamentos vizinhos concordam sobre a probabilidade posterior de quaisquer variáveis que eles tenham em comum. Com uma contabilidade cuidadosa, esse algoritmo é capaz de calcular probabilidades posteriores para todos os nós que não são de evidência na rede no tempo *linear* no tamanho da rede com agrupamentos. No entanto, a NP-dificuldade do problema não desapareceu: se uma rede exigir tempo e espaço exponenciais com a eliminação de variáveis, as TPCs na rede com agrupamentos necessariamente serão exponencialmente grandes.

14.5 INFERÊNCIA APROXIMADA EM REDES BAYESIANAS

Dada a intratabilidade da inferência exata em redes extensas com várias conexões, é essencial considerar métodos de inferência aproximada. Esta seção descreve algoritmos de amostragem aleatória, também chamados algoritmos de **Monte Carlo**, que fornecem respostas aproximadas cuja exatidão depende do número de amostras geradas. Em anos recentes, os algoritmos de Monte Carlo dos quais a têmpera simulada é um exemplo, são utilizados em muitas ramificações da ciência da computação para estimar quantidades que são difíceis de calcular com exatidão. Nesta seção, estamos interessados na amostragem aplicada à computação de probabilidades posteriores. Descreveremos duas famílias de algoritmos: amostragem direta e amostragem de cadeias de Markov. Duas outras abordagens — métodos variacionais e propagação com laços — serão mencionadas nas notas no final do capítulo.

14.5.1 Métodos de amostragem direta

O elemento primitivo em qualquer algoritmo de amostragem é a geração de amostras a partir de uma distribuição de probabilidade conhecida. Por exemplo, uma moeda imparcial pode ser considerada uma variável aleatória *Moeda* com os valores ⟨*cara*, *coroa*⟩ e uma distribuição *a priori* **P**(*Moeda*) = ⟨0,5, 0,5⟩. A amostragem a partir dessa distribuição é exatamente igual ao lançamento da moeda: com probabilidade 0,5 ela retornará *cara*, e com probabilidade 0,5 retornará *coroa*. Dada uma fonte de números aleatórios uniformemente distribuídos no intervalo [0, 1], é uma questão simples realizar a amostragem de qualquer distribuição sobre uma única variável, se discreta ou contínua (veja o Exercício 14.17).

A espécie mais simples de processo de amostragem aleatória para redes bayesianas gera eventos a partir de uma rede que não tem nenhuma evidência associada a ela. A ideia é fazer a amostragem uma variável de cada vez, em ordem topológica. A distribuição de probabilidade a partir da qual se obtém uma amostra do valor está condicionada aos valores já atribuídos aos pais da variável. Esse algoritmo é apresentado na Figura 14.13. Podemos ilustrar sua operação sobre a rede da Figura 14.12(a) supondo uma ordenação [*Nublado*, *Irrigador*, *Chuva*, *GramaMolhada*]:

função AMOSTRA-A-PRIORI(*rb*) **retorna** um evento amostrado a partir da probabilidade *a priori* especificada por *rb*

entradas: *rb,* uma rede bayesiana que especifica a distribuição conjunta $\mathbf{P}(X_1, ..., X_n)$

x ← um evento com *n* elementos
para cada variável X_i, **em** $X_1,....,X_n$ **faça**
  x[*i*] ← *u*ma amostra aleatória de $\mathbf{P}(X_i \mid pais(X_i))$
retornar x

Figura 14.13 Um algoritmo de amostragem que gera eventos de uma rede bayesiana. Cada variável é amostrada de acordo com a distribuição condicional dados os valores já amostrados para os pais da variável.

1. Amostra de **P**(*Nublado*) = ⟨0,5, 0,5⟩; valor é *verdadeiro*.
2. Amostra de **P**(*Irrigador* | *Nublado* = *verdadeiro*) = ⟨0,1, 0,9⟩; valor é *falso*.
3. Amostra de **P**(*Chuva* | *Nublado* = *verdadeiro*) = ⟨0,8, 0,2⟩; valor é *verdadeiro*.
4. Amostra de **P**(*GramaMolhada* | *Irrigador*, = *falso*, *Chuva* = *verdadeiro*) = ⟨0,9, 0,1⟩; valor é *verdadeiro*.

Nesse caso, AMOSTRA-*A-PRIORI* retorna o evento [*verdadeiro*, *falso*, *verdadeiro*, *verdadeiro*].

É fácil ver que AMOSTRA-A-PRIORI gera amostras a partir da distribuição conjunta *a priori* especificada pela rede. Primeiro, seja $S_{PS}(x_1, \ldots, x_n)$ a probabilidade de um evento específico ser gerado pelo algoritmo AMOSTRA-A-PRIORI. *Apenas observando o processo de amostragem*, temos:

$$S_{PS}(x_1 \ldots x_n) = \prod_{i=1}^{n} P(x_i \mid pais\ (X_i))$$

porque cada etapa de amostragem depende apenas dos valores dos pais. Essa expressão deve parecer familiar porque também é a probabilidade do evento de acordo com a representação da rede bayesiana da distribuição conjunta, conforme observamos na Equação 14.2. Isto é, temos:

$$S_{PS}(x_1 \ldots x_n) = P(x_1 \ldots x_n) .$$

Esse fato simples torna muito fácil responder a perguntas utilizando amostras.

Em qualquer algoritmo de amostragem, as respostas são calculadas efetuando-se a contagem das amostras reais geradas. Suponha que existam *N* amostras ao todo e seja $N_{PS}(x_1, \ldots, x_n)$ o número de vezes que o evento específico $x_1, \ldots, x_n$ ocorre no conjunto de amostras. Esperamos que esse número seja uma fração do total para convergir no limite para seu valor esperado de acordo com a probabilidade de amostragem:

$$\lim_{N\to\infty} \frac{N_{PS}(x_1, \ldots, x_n)}{N} = S_{PS}(x_1, \ldots, x_n) = P(x_1, \ldots, x_n) . \qquad (14.5)$$

Por exemplo, considere o evento produzido anteriormente: [*verdadeiro*, *falso*, *verdadeiro*,

verdadeiro]. A probabilidade de amostragem para esse evento é:

$$SPS(verdadeiro, falso, verdadeiro, verdadeiro) = 0,5 \times 0,9 \times 0,8 \times 0,9 = 0,324.$$

Consequentemente, no limite de N grande, esperamos que 32,4% das amostras sejam desse evento.

Sempre que usamos uma igualdade aproximada ("≈") no que se segue, queremos indicar exatamente esse sentido — que a probabilidade estimada se torna exata no limite de uma amostra grande. Tal estimativa é chamada **consistente**. Por exemplo, pode-se produzir uma estimativa consistente da probabilidade de qualquer evento parcialmente especificado, $x_1, \ldots, x_m$, onde $m \leq n$, como a seguir:

$$P(x_1, \ldots, x_m) \approx N_{PS}(x_1, \ldots, x_m)/N. \qquad (14.6)$$

Ou seja, a probabilidade do evento pode ser estimada como a fração de todos os eventos completos gerados pelo processo de amostragem que correspondem ao evento parcialmente especificado. Por exemplo, se gerarmos 1.000 amostras da rede de irrigadores e 511 delas tiverem *Chuva* = *verdadeiro*, então a probabilidade estimada de chuva, escrita como $\hat{P}(Chuva = verdadeiro)$, será 0,511.

Amostragem de rejeição em redes bayesianas

A **amostragem de rejeição** é um método geral para produzir amostras a partir de uma distribuição difícil de amostrar, dada uma distribuição fácil de amostrar. Em sua forma mais simples, ela pode ser usada para calcular probabilidades condicionais, isto é, para determinar $P(X \mid \mathbf{e})$. O algoritmo AMOSTRAGEM-DE-REJEIÇÃO é representado na Figura 14.14. Primeiro, ele gera amostras a partir da distribuição *a priori* especificada pela rede. Em seguida, rejeita todas as que não correspondem à evidência. Finalmente, a estimativa $\hat{P}(X = x \mid \mathbf{e})$ é obtida pela contagem da frequência com que $X = x$ ocorre nas amostras restantes.

```
função AMOSTRAGEM-DE-REJEIÇÃO(X, e, rb, N) retorna uma estimativa de P(X | e)
  entradas: X, a variável de consulta
            e, valores observados para variáveis E
            rb, uma rede bayesiana
            N, o número total de amostras a serem geradas
  variáveis locais: N, um vetor de contagens para cada valor de X, inicialmente zero
  para j = 1 até N faça
    x ← AMOSTRA-A-PRIORI(rb)
    se x é consistente com e então
      N[x] ← N[x]+1 onde x é o valor de X em x
  retornar NORMALIZAR(N)
```

Figura 14.14 O algoritmo de amostragem de rejeição para responder a consultas, dada a evidência em uma rede bayesiana.

Seja $\hat{\mathbf{P}}(X \mid \mathbf{e})$ a distribuição estimada que o algoritmo retorna. A partir da definição do algoritmo, temos:

$$\hat{\mathbf{P}}(X \mid \mathbf{e}) = \alpha \mathbf{N}_{PS}(X, \mathbf{e}) = \frac{\mathbf{N}_{PS}(X, \mathbf{e})}{N_{PS}(\mathbf{e})} .$$

A partir da Equação 14.6, isso se transforma em:

$$\hat{\mathbf{P}}(X \mid \mathbf{e}) \approx \frac{\mathbf{P}(X, \mathbf{e})}{P(\mathbf{e})} = \mathbf{P}(X \mid \mathbf{e}) .$$

Ou seja, a amostragem de rejeição produz uma estimativa consistente da probabilidade verdadeira.

Continuando com nosso exemplo da Figura 14.12(a), vamos supor que desejamos estimar **P**(*Chuva*|*Irrigador* = *verdadeiro*), utilizando 100 amostras. Das 100 amostras que geramos, suponha que 73 tenham *Irrigador* = *falso* e sejam rejeitadas, enquanto 27 têm *Irrigador* = *verdadeiro*; destas, 27, 8 têm *Chuva* = *verdadeiro* e 19 têm *Chuva* = *falso*. Consequentemente,

$$\mathbf{P}(Chuva \mid Irrigador = verdadeiro) \approx \text{NORMALIZAR}(\langle 8, 19 \rangle) = \langle 0{,}296,\ 0{,}704 \rangle.$$

A resposta verdadeira é ⟨0,3, 0,7⟩. À medida que mais amostras forem coletadas, a estimativa convergirá para a resposta verdadeira. O desvio-padrão do erro em cada probabilidade será proporcional a $1/\sqrt{n}$, onde *n* é o número de amostras usadas na estimativa.

O maior problema com a amostragem de rejeição é que ela rejeita muitas amostras! A fração de amostras consistentes com a evidência **e** cai exponencialmente conforme o número de variáveis de evidência cresce e, assim, o procedimento é simplesmente inútil para problemas complexos.

Note que a amostragem de rejeição é muito semelhante à avaliação de probabilidades condicionais diretamente do mundo real. Por exemplo, para estimar **P**(*Chuva* | *CéuVermelhoNoite* = *verdadeiro*), pode-se simplesmente contar com que frequência chove depois que se observa um céu vermelho na noite anterior — ignorando-se as noites em que o céu não está vermelho (aqui, o próprio mundo desempenha o papel do algoritmo de geração de amostras). É óbvio que isso poderia tomar um longo tempo, se o céu só muito raramente ficasse vermelho, e essa é a deficiência da amostragem de rejeição.

Ponderação de probabilidade

A **ponderação de probabilidade** evita a ineficiência da amostragem de rejeição gerando apenas eventos consistentes com a evidência **e**. É uma instância particular da técnica estatística geral de **amostragem de importância**, sob medida para inferência em redes bayesianas. Começamos descrevendo como o algoritmo funciona; em seguida, mostraremos que ele funciona corretamente, isto é, gera estimativas de probabilidade consistentes.

A PONDERAÇÃO-DE-PROBABILIDADE (veja a Figura 14.15) fixa os valores para as variáveis de evidência **E** e amostras apenas de variáveis ocultas. Isso garante que cada evento gerado será consistente com a evidência. Porém, nem todos os eventos são iguais. Antes de efetuar as contas na distribuição para a variável de consulta, cada evento é ponderado pela *probabilidade* de que o

evento concorde com a evidência, medida pelo produto das probabilidades condicionais para cada variável de evidência, dados seus pais. Intuitivamente, eventos em que a evidência real parece improvável devem receber menor peso.

```
função PONDERAÇÃO-DE-PROBABILIDADE (X, e, rb, N) retorna uma estimativa de P(X | e)
  entradas: X, a variável de consulta
            e, valores observados para variáveis E
            rb, uma rede bayesiana especificando distribuição conjunta P(X1,....,Xn)
            N, o número total de amostras a serem geradas
  variáveis locais: W, um vetor de contagens ponderadas para cada valor de X, inicialmente igual a
  para j = 1 até N faça
    x, w ← AMOSTRA-PONDERADA(rb, e)
    W[x] ← W[x] + w onde x é o valor de X em x
  retornar NORMALIZAR(W)
```

```
função AMOSTRA-PONDERADA(rb, e) retorna um evento e um peso
  w ← 1; x ← um evento com n elementos inicializados de e
  para cada variável Xi em X1,....,Xn faça
    se Xi é uma variável de evidência com valor xi em e
      então w ← w × P(Xi = xi | pais(Xi))
      else x[i]←uma amostra aleatória de P(Xi | pais(Xi))
  retornar x, w
```

Figura 14.15 Algoritmo de ponderação de probabilidade para inferência em redes bayesianas. Em AMOSTRA-PONDERADA, cada variável oculta é amostrada de acordo com a distribuição condicional, dados os valores já amostrados para os pais da variável, enquanto um peso é acumulado baseado na probabilidade para cada variável de evidência.

Vamos aplicar o algoritmo à rede apresentada na Figura 14.12(a), com a consulta **P**(*Chuva* | *Nublado* = *verdadeiro*, *GramaMolhada* = *verdadeiro*) e a ordenação *Nublado, Irrigador,Chuva,GramaMolhada* (qualquer ordem topológica vai funcionar). O processo se desenvolve assim: primeiro, o peso *w* é definido como 1.0. Em seguida, é gerado um evento:

1. *Nublado* é uma variável de evidência com valor *verdadeiro*. Portanto, vamos definir

   $w \leftarrow w \times P(Nublado = verdadeiro) = 0{,}5$.
2. *Irrigador* não é uma variável de evidência, então a amostra de **P**(*Irrigador* | *Nublado* = *verdadeiro*) = ⟨0,1, 0,9⟩; suponha que retorne *falso*.
3. Da mesma forma, amostra de **P**(*Chuva* | *Nublado* = *verdadeiro*) = ⟨0,8, 0,2⟩; suponha que retorne verdadeiro.
4. *GramaMolhada* é uma variável de evidência com valor *verdadeiro*. Portanto, definimos

$$w \leftarrow w \times P(GramaMolhada = verdadeiro \mid Irrigador = falso, Chuva = verdadeiro) = 0{,}45.$$

Nesse caso, AMOSTRA-PONDERADA retorna o evento [*verdadeiro*, *verdadeiro*, *verdadeiro*, *verdadeiro*] com peso 0,45, e isso é registrado sob *Chuva* = *verdadeiro*.

Para entender por que a ponderação de probabilidade funciona, começamos examinando a distribuição de amostragem S_{WS} para AMOSTRA-PONDERADA. Lembre-se de que as variáveis de evidência **E** são fixadas com valores **e**. Chamamos de variáveis **Z** que não são de evidência (inclusive a variável de consulta *X*). O algoritmo realiza a amostragem de cada variável em **Z** dados os valores de seus pais:

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i \mid pais(Z_i)) \,. \tag{14.7}$$

Note que $Pais(Z_i)$ pode incluir ambas, variáveis ocultas e de evidência. Diferentemente da distribuição *a priori* $P(\mathbf{z})$, a distribuição S_{WS} dedica alguma atenção à evidência: os valores amostrados para cada Z_i serão influenciados pela evidência entre os ancestrais de Z_i. Por exemplo, quando é feita a amostra de *Irrigador*, o algoritmo presta atenção à evidência *Nublado* = *verdadeiro* na sua variável pai. Por outro lado, S_{WS} dedica menor atenção à evidência do que a distribuição posterior verdadeira $P(\mathbf{z} \mid \mathbf{e})$ porque os valores amostrados para cada Z_i *ignoram* a evidência entre os que não são ancestrais de Z_i.[5] Por exemplo, quando se tira amostra de *Irrigador* e *Chuva*, o algoritmo ignora a evidência na variável filho *GramaMolhada* = *verdadeiro*; isso significa que vai gerar muitas amostras com *Irrigador* = *falso e Chuva* = *falso* apesar do fato de que a evidência na realidade descarta esse caso.

O peso de probabilidade *w* constitui a diferença entre as distribuições de amostragem real e desejada. O peso para dada amostra **x**, composta de **z** e **e**, é o produto das probabilidades para cada variável de evidência, dados seus pais (dos quais alguns ou todos podem estar entre os valores Z_i):

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i \mid pais(E_i)) \,. \tag{14.8}$$

Multiplicando as Equações 14.7 e 14.8, vemos que a probabilidade *ponderada* de uma amostra tem a forma particularmente conveniente:

$$\begin{aligned} S_{WS}(\mathbf{z}, \mathbf{e}) w(\mathbf{z}, \mathbf{e}) &= \prod_{i=1}^{l} P(z_i \mid pais\,(Z_i)) \prod_{i=1}^{m} P(e_i \mid pais\,(E_i)) \\ &= P(\mathbf{z}, \mathbf{e}) \end{aligned} \tag{14.9}$$

porque os dois produtos abrangem todas as variáveis da rede, permitindo-nos utilizar a Equação 14.2 para a probabilidade conjunta.

Agora é fácil mostrar que as estimativas de ponderação de probabilidade são consistentes. Para qualquer valor específico *x* de *X*, a probabilidade posterior estimada pode ser calculada como:

$$\begin{aligned}
\hat{P}(x \mid \mathbf{e}) &= \alpha \sum_{\mathbf{y}} N_{WS}(x, \mathbf{y}, \mathbf{e}) w(x, \mathbf{y}, \mathbf{e}) && \text{a partir de PONDERAÇÃO-DE-PROBABILIDADE} \\
&\approx \alpha' \sum_{\mathbf{y}} S_{WS}(x, \mathbf{y}, \mathbf{e}) w(x, \mathbf{y}, \mathbf{e}) && \text{para } N \text{ grande} \\
&= \alpha' \sum_{\mathbf{y}} P(x, \mathbf{y}, \mathbf{e}) && \text{pela Equação 14.9} \\
&= \alpha' P(x, \mathbf{e}) = P(x \mid \mathbf{e}) .
\end{aligned}$$

Consequentemente, a ponderação de probabilidade retorna estimativas consistentes.

Tendo em vista que a ponderação de probabilidade utiliza todas as amostras geradas, ela pode ser muito mais eficiente que a amostragem de rejeição. Entretanto, ela sofrerá uma degradação de desempenho à medida que o número de variáveis de evidência aumentar. Como muitas amostras terão pesos muito baixos, consequentemente a estimativa ponderada será dominada pela minúscula fração de amostras que concordam em uma proporção maior que uma probabilidade infinitesimal com a evidência. O problema será exacerbado se as variáveis de evidência ocorrerem mais tardiamente na ordenação das variáveis porque as variáveis ocultas não terão evidência em seus pais e ancestrais para guiar a geração de amostras. Isso significa que as amostras serão simulações que terão pouca semelhança com a realidade sugerida pela evidência.

14.5.2 Inferência por simulação de cadeias de Markov

Os algoritmos de **Monte Carlo via cadeia de Markov** (CMMC) trabalham de forma bastante diferente da amostragem de rejeição e da ponderação de probabilidade. Em vez de gerar cada amostra a partir do zero, os algoritmos CMMC geram cada amostra, fazendo uma mudança aleatória na amostra anterior. É útil, portanto, pensar em um algoritmo CMMC como estando em determinado *estado atual* especificando um valor para cada variável e gerando um *estado próximo*, fazendo mudanças aleatórias no estado atual (se isso o faz lembrar a têmpera simulada do Capítulo 4 ou o WALKSAT do Capítulo 7, deve-se ao fato de ambos serem membros da família CMMC). Descrevemos aqui uma forma especial do CMMC chamada **amostragem de Gibbs**, que é especialmente adequada para as redes bayesianas (outras formas, algumas delas muito mais poderosas, serão discutidas nas notas ao final do capítulo). Vamos primeiro descrever o que o algoritmo faz; depois explicaremos por que funciona.

A amostragem de Gibbs em redes bayesianas

O algoritmo de amostragem de Gibbs para redes bayesianas começa com um estado arbitrário (com as variáveis de evidência fixadas em seus valores observados) e gera um estado próximo por amostrar aleatoriamente um valor para uma das variáveis ocultas X_i. A amostragem para X_i é feita *condicionada sobre os valores atuais das variáveis na cobertura de Markov de* X_i (lembre-se de que já vimos que a cobertura de Markov de uma variável consiste em seus pais, filhos e pais dos filhos). Então, o algoritmo vagueia ao acaso pelo espaço de estados — o espaço de atribuições completas possíveis, invertendo uma variável de cada vez, mas mantendo fixas as variáveis de evidência.

Considere a consulta **P**(*Chuva* | *Irrigador* = *verdadeiro*, *GramaMolhada* = *verdadeiro*) aplicada à rede na Figura 14.12(a). As variáveis de evidência *Irrigador* e *GramaMolhada* são fixadas com seus valores observados e as variáveis ocultas *Nublado* e *Chuva* são inicializadas aleatoriamente — digamos que elas sejam inicializadas como *verdadeiro* e *falso*, respectivamente. Desse modo, o estado inicial é [*verdadeiro*, *verdadeiro*, *falso*, *verdadeiro*]. Agora as etapas a seguir são executadas repetidamente:

1. *Nublado* é amostrada, dados os valores atuais de suas variáveis de cobertura de Markov: nesse caso, obtemos a amostra a partir de **P**(*Nublado* | *Irrigador* = *verdadeiro*, *Chuva* = *falso*) (em breve, mostraremos como calcular essa distribuição). Suponha que o resultado seja *Nublado* = *falso*. Então, o novo estado atual é [*falso*, *verdadeiro*, *falso*, *verdadeiro*].
2. *Chuva* é amostrada, dados os valores atuais de suas variáveis de cobertura de Markov: nesse caso, obtemos a amostra a partir de **P**(*Chuva* | *Nublado* = *falso*, *Irrigador* = *verdadeiro*, *GramaMolhada* = *verdadeiro*). Suponha que isso produza *Chuva* = *verdadeiro*. O novo estado atual é [*falso*, *verdadeiro*, *verdadeiro*, *verdadeiro*].

Cada estado visitado durante esse processo é uma amostra que contribui para a estimativa referente à variável de consulta *Chuva*. Se o processo visitar 20 estados em que *Chuva* tem valor verdadeiro e 60 estados em que *Chuva* tem valor falso, a resposta à consulta será NORMALIZAR(⟨20, 60⟩) = ⟨0,25, 0,75⟩. O algoritmo completo é mostrado na Figura 14.16.

```
função ASK-GIBBS(X, e, rb, N) retorna uma estimativa de P(X | e)
  variáveis locais: N, um vetor de contagens sobre X, inicialmente zero
                    Z, as variáveis não de evidência em rb
                    x, o estado atual da rede, inicialmente copiado de e

  inicializar x com valores aleatórios para as variáveis em Z
  para j = 1 até N faça
    para cada Z_i em Z faça
      fazer a amostragem do valor de Z_i em x a partir de P(Z_i | mb(Z_i))
      SALTO, V.O. 537
  retornar NORMALIZAR(N)
```

Figura 14.16 Algoritmo de amostragem de Gibbs para inferência aproximada em redes bayesianas; essa versão passa por um ciclo através das variáveis, mas a escolha de variáveis aleatórias também funciona.

Por que o algoritmo de Gibbs funciona

Agora, mostraremos que a amostragem de Gibbs retorna estimativas consistentes para probabilidades posteriores. O material desta seção é bastante técnico, mas a afirmação básica é simples: *o processo de amostragem se fundamenta em um "equilíbrio dinâmico" no qual a fração a longo prazo do tempo gasto em cada estado é exatamente proporcional à sua probabilidade*

posterior. Essa notável propriedade decorre da **probabilidade de transição** específica com que o processo passa de um estado para outro, definida pela distribuição condicional dada pela cobertura de Markov da variável cuja amostra está sendo coletada.

Seja $\theta(\mathbf{x}, \rightarrow \mathbf{x}')$ a probabilidade de que o processo faça uma transição do estado **x** para o estado **x**′. Essa probabilidade de transição define o que se denomina **cadeia de Markov** sobre o espaço de estados (as cadeias de Markov também se destacarão nos Capítulos 15 e 17). Agora, suponha que executemos a cadeia de Markov para t etapas, e seja $\pi_t(\mathbf{x})$ a probabilidade de que o sistema esteja no estado **x** no tempo t. De modo semelhante, seja $\pi_{t+1}(\mathbf{x}')$ a probabilidade de o sistema se encontrar no estado **x**′ no tempo $t + 1$. Dado $\pi_t(\mathbf{x})$, podemos calcular $\pi_{t+1}(\mathbf{x}')$ efetuando o somatório da probabilidade de estar em um estado multiplicada pela probabilidade de fazer a transição para **x**′, para todos os estados em que o sistema poderia se encontrar no tempo t:

$$\pi_{t+1}(\mathbf{x}') = \sum_{\mathbf{x}} \pi_t(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \ .$$

Diremos que a cadeia alcançou sua **distribuição estacionária** se $\pi_t = \pi_{t+1}$. Vamos chamar essa distribuição estacionária de π; sua equação de definição é portanto:

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \quad \text{para todo } \mathbf{x}' \ . \qquad (14.10)$$

Desde que a distribuição de probabilidade de transição θ seja **ergódica** — isto é, cada estado pode ser alcançado de todos os outros e não há ciclos rigorosamente periódicos —, existe exatamente uma distribuição π que satisfaz essa equação para qualquer θ dado.

A Equação 14.10 pode ser interpretada com o significado de que o "fluxo de saída" esperado a partir de cada estado (isto é, sua "população" atual) é igual ao "fluxo de entrada" de todos os estados. Um modo óbvio de satisfazer esse relacionamento ocorre se o fluxo esperado entre qualquer par de estados é o mesmo em ambos os sentidos, isto é,

$$\pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) \quad \text{para todo } \mathbf{x}, \ \mathbf{x}' \ . \qquad (14.11)$$

Quando essas equações são válidas podemos dizer que $\theta(\mathbf{x} \rightarrow \mathbf{x}')$ está em **equilíbrio detalhado** com $\pi(\mathbf{x})$.

Podemos mostrar que o equilíbrio detalhado implica imutabilidade simplesmente efetuando o somatório sobre **x** na Equação 14.11. Temos:

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}') \sum_{\mathbf{x}} q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}')$$

onde a última etapa se segue porque temos a garantia de que vai ocorrer uma transição a partir de **x**′.

Agora, mostraremos que a probabilidade de transição $\theta(\mathbf{x}, \mathbf{x}')$ definida pela etapa de amostragem em GIBBS-ASK na verdade é um caso especial da definição mais geral da amostragem de Gibbs, segundo a qual cada variável é amostrada condicionalmente sobre os valores atuais de *todas* as outras variáveis. Começamos por mostrar que essa definição geral da amostragem de Gibbs satisfaz

a equação de balanço detalhado com uma distribuição estacionária igual a $P(\mathbf{x} \mid \mathbf{e})$, (a distribuição posterior verdadeira sobre variáveis ocultas). Então, observamos simplesmente que, para redes bayesianas, a amostragem condicionalmente em todas as variáveis é equivalente a fazer a amostragem condicional sobre a cobertura de Markov da variável.

Para analisar o amostrador geral de Gibbs, que faz a amostra de cada X_i por vez, com uma probabilidade de transição θ_i que impõe condições sobre todas as outras variáveis, definimos $\overline{\mathbf{X}_i}$ como essas outras variáveis (exceto as variáveis de evidência); seus valores no estado atual são $\overline{\mathbf{x}_i}$. Se fizermos a amostragem de um novo valor x'_i para X_i condicionalmente sobre todas as outras variáveis, inclusive a evidência, teremos:

$$q_i(\mathbf{x} \to \mathbf{x}') = q_i((x_i, \overline{\mathbf{x}_i}) \to (x'_i, \overline{\mathbf{x}_i})) = P(x'_i \mid \overline{\mathbf{x}_i}, \mathbf{e}) .$$

Agora mostraremos que a probabilidade de transição para cada etapa da amostragem de Gibbs está em equilíbrio detalhado com a distribuição posterior verdadeira:

$$\begin{aligned}
\pi(\mathbf{x})q_i(\mathbf{x} \to \mathbf{x}') &= P(\mathbf{x} \mid \mathbf{e})P(x'_i \mid \overline{\mathbf{x}_i}, \mathbf{e}) = P(x_i, \overline{\mathbf{x}_i} \mid \mathbf{e})P(x'_i \mid \overline{\mathbf{x}_i}, \mathbf{e}) \\
&= P(x_i \mid \overline{\mathbf{x}_i}, \mathbf{e})P(\overline{\mathbf{x}_i} \mid \mathbf{e})P(x'_i \mid \overline{\mathbf{x}_i}, \mathbf{e}) && \text{(usando a regra da cadeia no primeiro termo)} \\
&= P(x_i \mid \overline{\mathbf{x}_i}, \mathbf{e})P(x'_i, \overline{\mathbf{x}_i} \mid \mathbf{e}) && \text{(usando a regra da cadeia no sentido inverso)} \\
&= \pi(\mathbf{x}')q_i(\mathbf{x}' \to \mathbf{x}) .
\end{aligned}$$

Podemos pensar no laço **"para cada** Z_i **em Z faça"** da Figura 14.16 como definindo uma grande probabilidade de transição θ que é a composição sequencial $\theta_1 \circ \theta_2 \circ \ldots \circ \theta_n$ das probabilidades de transição para as variáveis individuais. É fácil mostrar (Exercício 14.19) que, se cada θ_i e θ_j tiver π como distribuição estacionária, a composição sequencial $\theta_i \circ \theta_j$ também terá, daí a probabilidade de transição θ para o laço todo terá $P(\mathbf{x} \mid \mathbf{e})$ como sua distribuição estacionária. Finalmente, a menos que os TPCs contenham probabilidades de 0 ou 1 — que pode fazer com que o espaço de estado se torne desconectado — é fácil verificar que θ é ergódica. Assim, as amostras geradas pela amostragem de Gibbs acabarão por ser extraídas eventualmente pela distribuição posterior verdadeira.

A etapa final é mostrar como realizar a etapa de amostragem geral de Gibbs geral — amostragem X_i de $\mathbf{P}(X_i \mid \overline{\mathbf{X}_i}, e)$ — em uma rede bayesiana. Lembre-se que uma variável é independente de todas as outras variáveis dada a cobertura de Markov; daí,

$$P(x'_i \mid \overline{\mathbf{x}_i}, \mathbf{e}) = P(x'_i \mid mb(X_i)) ,$$

onde $mb(X_i)$ denota os valores das variáveis na cobertura de Markov X_i, $MB(X_i)$. Como mostrado no Exercício 14.7, a probabilidade de uma variável dada sua cobertura de Markov é proporcional à probabilidade da variável dados seus pais vezes a probabilidade de cada filho dados seus pais respectivos:

$$P(x_i' \mid mb(X_i)) = \alpha\, P(x_i' \mid pais(X_i)) \times \prod_{Y_j \in filhos(X_i)} P(y_j \mid pais(Y_j))\,. \qquad (14.12)$$

Daí, para tornar cada variável X_i condicionada à sua cobertura de Markov, o número de multiplicações necessárias é igual ao número de filhos de X_i.

14.6 MODELOS DE PROBABILIDADE RELACIONAL E DE PRIMEIRA ORDEM

No Capítulo 8, explicamos as vantagens de representação apresentadas pela lógica de primeira ordem em comparação à lógica proposicional. A lógica de primeira ordem se refere à existência de objetos e relações entre eles, e pode expressar fatos sobre *alguns* ou *todos* os objetos de um domínio. Com frequência, isso resulta em representações muito mais concisas que as descrições proposicionais equivalentes. As redes bayesianas são essencialmente proposicionais: o conjunto de variáveis aleatórias é fixo e finito, e cada um tem um domínio fixo de valores possíveis. Esse fato limita a aplicabilidade das redes bayesianas. *Se pudermos encontrar um modo de combinar a teoria da probabilidade com o poder de expressão das representações de primeira ordem*, *esperamos poder aumentar drasticamente a variedade de problemas que podem ser tratados*.

Por exemplo, suponha que um varejista de livros on-line gostaria de fornecer avaliações globais de produtos com base nas recomendações recebidas de seus clientes. Dada a evidência disponível, a avaliação assumirá a forma de distribuição posterior sobre a qualidade do livro. É a solução mais simples para basear a avaliação sobre a recomendação média, talvez com uma variação determinada pela quantidade de recomendações, mas sem levar em conta o fato de que alguns clientes são mais amáveis do que os outros e alguns são menos honestos do que outros. Os clientes gentis tendem a dar boas recomendações, mesmo para livros bastante medíocres, enquanto os clientes desonestos dão recomendações muito boas ou muito ruins por outras razões que não a qualidade — por exemplo, podem trabalhar para um editor.[6]

Para um único cliente C_1, recomendando um único livro B_1, a rede de Bayes pode parecer como a mostrada na Figura 14.17(a). (Assim como na Seção 9.1, as expressões com parênteses como Honesto(C_1) são apenas símbolos de fantasia — nesse caso, nomes de fantasia para variáveis aleatórias.)

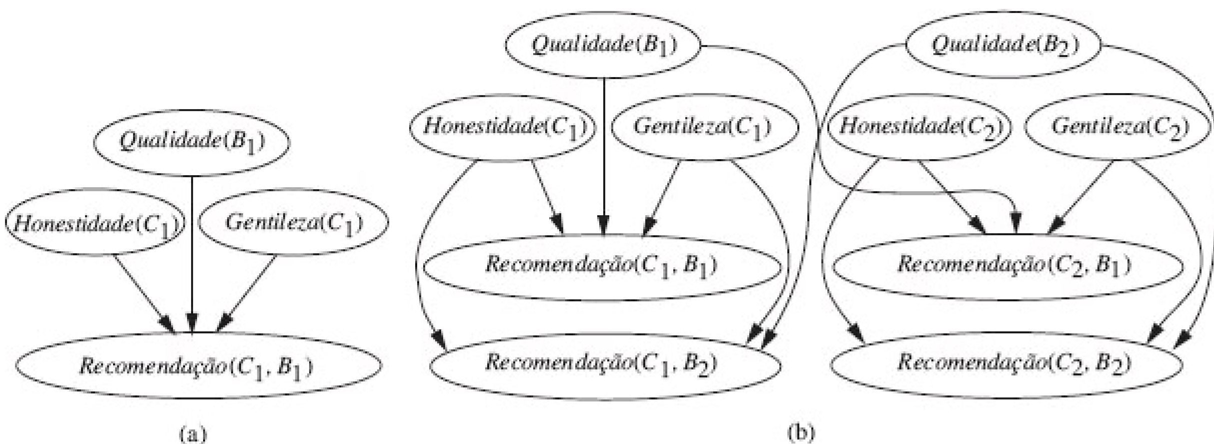


Figura 14.17 (a) Rede de Bayes para um único cliente C_1 recomendando um único livro B_1. *Honesto*(C_1) é booleana, enquanto as outras variáveis têm valores inteiros de 1 a 5. (b) Rede de Bayes com dois clientes e dois livros.

Com dois clientes e dois livros, a rede de Bayes se parece com a da Figura 14.17(b). Para quantidades maiores de livros e clientes, torna-se completamente inviável especificar a rede manualmente.

Felizmente, a rede tem muita estrutura repetida. Cada variável *Recomendação*(*c*, *b*) tem as variáveis *Honesto*(*c*), *Gentileza*(*c*) e *Qualidade*(*b*) como seus pais. Além disso, os TPCs de todas as variáveis *Recomendação*(*c*, *b*) são idênticos, como o são aqueles de todas as variáveis *Honesto*(*c*), e assim por diante. A situação parece feita sob medida para uma linguagem de primeira ordem. Gostaríamos de dizer algo como

$$Recomendação(c, b) \sim RecTPC(Honesto(c), Gentileza(c), Qualidade(b))$$

com o significado pretendido que a recomendação de um cliente para um livro depende da honestidade e gentileza do cliente e da qualidade do livro de acordo com alguns TPCs fixos. Esta seção desenvolve uma linguagem que nos permite dizer exatamente isso e muito mais.

14.6.1 Mundos possíveis

Lembre-se do Capítulo 13, que um modelo de probabilidade define um conjunto Ω de mundos possíveis com probabilidade *P*(w) para cada mundo w. Para redes bayesianas, os mundos possíveis são atribuições de valores para as variáveis; para o caso booleano em particular, os mundos possíveis são idênticos aos da lógica proposicional. Para o modelo de probabilidade de primeira ordem, então, parece que precisamos de mundos possíveis como aqueles da lógica de primeira ordem, isto é, um conjunto de objetos com relações entre eles e uma interpretação que mapeia símbolos constantes para objetos, símbolos predicados para relações e símbolos de função para funções sobre esses objetos (veja a Seção 8.2). O modelo também precisa definir uma probabilidade para cada mundo possível, tal como uma rede bayesiana define uma probabilidade para cada atribuição de valores para as variáveis.

Vamos supor, por um momento, que descobrimos como fazer isso. Então, como de costume, podemos obter a probabilidade de qualquer sentença lógica de primeira ordem ϕ como uma soma de mundos possíveis onde ele é verdadeiro:

$$P(\phi) = \sum_{\omega:\phi \text{ é verdadeiro em } \omega} P(\omega) . \qquad (14.13)$$

Pode-se obter probabilidades condicionais $P(\phi \mid \mathbf{e})$ de forma semelhante; assim, podemos, em princípio, fazer qualquer pergunta que quisermos sobre o nosso modelo — por exemplo: “Quais livros são mais propensos a ser altamente recomendados por clientes desonestos?” — e obter uma resposta. Até agora, tudo bem.

Há, no entanto, um problema: o conjunto de modelos de primeira ordem é infinito. Vimos isso explicitamente na Figura 8.4, o que mostraremos novamente na Figura 14.18 (parte superior). Isso significa (1) que o somatório da Equação 14.13 pode ser inviável e (2) especificar distribuição consistente e completa sobre um conjunto infinito de mundos poderia ser muito difícil.

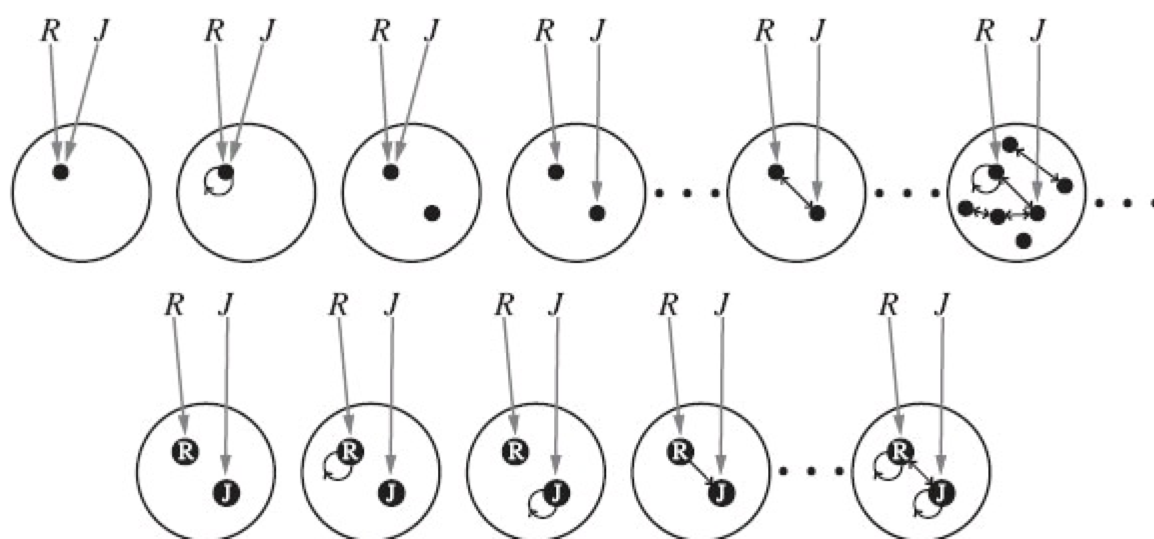


Figura 14.18 Em cima: alguns membros do conjunto de todos os mundos possíveis para uma linguagem com dois símbolos constantes, *R* e *J*, e um símbolo de relação binária, sob a semântica-padrão da lógica de primeira ordem. Embaixo: os mundos possíveis sob a semântica de banco de dados. A interpretação dos símbolos constantes é fixa, e há um objeto distinto para cada símbolo constante.

A Seção 14.6.2 explora uma abordagem para lidar com esse problema. A ideia é não adotar a semântica-padrão de lógica de primeira ordem, mas a **semântica do banco de dados** definida na Seção 8.2.8. A semântica de banco de dados assume a **hipótese de nomes únicos** — aqui, nós a adotamos para símbolos constantes; ela também assume o **fecho do domínio** — não há mais objetos do que aqueles que recebem nomes. Podemos, então, garantir um conjunto finito de mundos possíveis fazendo com que o conjunto de objetos em cada mundo seja exatamente o conjunto de símbolos constantes que são usados. Como mostrado na Figura 14.18 (parte inferior), não há incerteza sobre o mapeamento de símbolos a objetos ou sobre os objetos que existem. Chamamos os modelos definidos dessa forma de **modelos de probabilidade relacional**, ou MPRs.[7] A diferença mais significativa entre a semântica de MPRs e a semântica de banco de dados apresentada na Seção 8.2.8 é que MPRs não constroem o pressuposto de mundo fechado — obviamente, supor que cada fato desconhecido seja falso não faz sentido em um sistema de raciocínio probabilístico!

Quando os pressupostos subjacentes da semântica do banco de dados não conseguem se manter, os MPRs não funcionam bem. Por exemplo, um varejista de livro pode usar um ISBN (International Standard Book Number) como símbolo constante para denominar cada livro, mesmo que determinado livro "lógico" (por exemplo, *E o vento levou*) possa ter vários ISBNs. Faria sentido agregar recomendações através de múltiplos ISBNs, mas o varejista pode não saber com certeza quais ISBNs são realmente o mesmo livro (note que não estamos reificando as *cópias individuais* do livro, o que pode ser necessário para as vendas de livros usados, vendas de carros, e assim por diante). Pior ainda, cada cliente é identificado por um ID de login, mas um cliente desonesto pode ter milhares de IDs! No campo de segurança computacional, esses múltiplos Ids são chamados **sibilas**, e seu uso para

confundir um sistema de reputação é chamado de **ataque sibila**. Assim, mesmo uma aplicação simples em um domínio on-line relativamente bem definido envolve tanto a **incerteza da existência** (que são os verdadeiros livros e clientes subjacentes aos dados observados) e a **incerteza de identidade** (cujo símbolo na verdade se refere ao mesmo objeto). É necessário encarar e definir modelos de probabilidade com base na semântica-padrão de lógica de primeira ordem, para a qual os mundos possíveis variam nos objetos que contêm e nos mapeamentos de símbolos a objetos. A Seção 14.6.3 mostra como fazer isso.

14.6.2 Modelos de probabilidade relacional

Como a lógica de primeira ordem, os MPRs têm símbolos de constantes, funções e predicados (é mais fácil visualizar predicados como funções que retornam verdadeiro ou falso). Vamos também assumir uma **assinatura de tipos** para cada função, ou seja, uma especificação do tipo de cada argumento e do valor da função. Se o tipo de cada objeto for conhecido, muitos mundos possíveis espúrios serão eliminados por esse mecanismo. Para o domínio recomendado pelo livro, os tipos são *Cliente e Livro*, e as assinaturas dos tipos para as funções e predicados são as seguintes:

$Honesto: Cliente \rightarrow \{verdadeiro, falso\} Gentileza : Cliente \rightarrow \{1, 2, 3, 4, 5\}$
$Qualidade : Livro \rightarrow \{1, 2, 3, 4, 5\}$
$Recomendação : Cliente \times Livro \rightarrow \{1, 2, 3, 4, 5\}$

Os símbolos constantes serão os nomes dos clientes e livros que aparecem no conjunto dos dados do varejista. No exemplo dado anteriormente (Figura 14.17(b)), são C_1, C_2 e B_1, B_2.

Dadas as constantes e seus tipos, juntamente com as funções e seus tipos de assinaturas, as variáveis aleatórias do MPR são obtidas pela instanciação de cada função com cada combinação possível de objetos: $Honest\ (C_1)$, $Qualidade\ (B_2)$, $Recomendação\ (C_1, B_2)$, e assim por diante. Essas são exatamente as variáveis que aparecem na Figura 14.17(b). Como cada tipo tem apenas um número finito de instâncias, o número de variáveis aleatórias básicas também é finito.

Para completar o MPR, temos que escrever as dependências que governam essas variáveis aleatórias. Há apenas uma declaração de dependência para cada função, onde cada argumento da função é uma variável lógica (ou seja, uma variável que varia dentro dos limites dos objetos, como na lógica de primeira ordem):

$Honesto(c) \sim \langle 0{,}99, 0{,}01 \rangle$
$Gentileza(\mathrm{c}) \sim \langle 0{,}1, 0{,}1, 0{,}2, 0{,}3, 0{,}3 \rangle$
$Qualidade(\mathrm{b}) \sim \langle 0{,}05, 0{,}2, 0{,}4, 0{,}2, 0{,}15 \rangle$
$Recomendação(c, b) \sim RecTPC\ (Honesto(c), Gentileza(c), Qualidade(b))$

onde *RecTPC* é uma distribuição condicional definida separadamente com $2 \times 5 \times 5 = 50$ linhas, cada uma com cinco entradas. A semântica de MPR pode ser obtida pela instanciação dessas dependências de todas as constantes conhecidas, resultando em uma rede bayesiana (como na Figura 14.17(b)) que define uma distribuição conjunta sobre as variáveis aleatórias MPR.[8]

Podemos refinar o modelo, introduzindo uma **independência específica de contexto** para refletir o fato de que os clientes desonestos ignoram a qualidade ao dar uma recomendação; além disso, a gentileza não desempenha nenhum papel em suas decisões. A independência específica de contexto permite que uma variável seja independente de alguns de seus pais dado certos valores das outras; portanto, *Recomendação*(c, b) é independente de *Gentileza*(c) e de *Qualidade*(b) quando *Honesto*(c) = *falso:*

Recomendação(c, b)~ **se** *Honesto*(c) **então**
HonestoRecTPC(*Gentileza*(c), *Qualidade*(b))
senão ⟨0,4, 0,1, 0,0, 0,1, 0,4⟩.

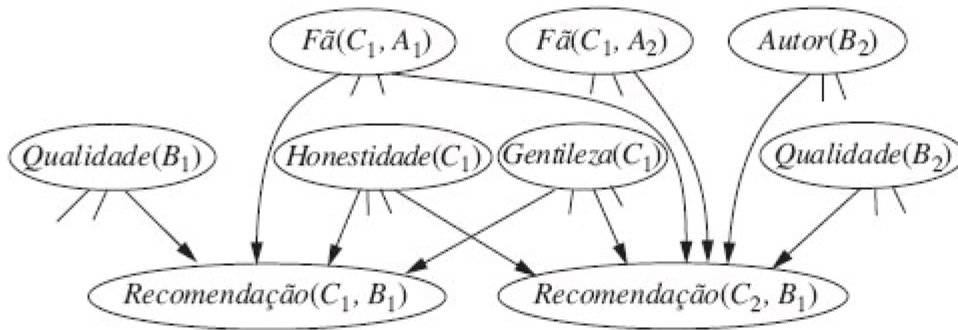


Figura 14.19 Fragmento da rede de Bayes equivalente quando *Autor*(B_2) é desconhecido.

Esse tipo de dependência pode ser parecido com uma instrução simples se-então-senão em uma linguagem de programação, mas há uma diferença fundamental: o mecanismo de *inferência não sabe necessariamente o valor do teste condicional!*

Podemos elaborar esse modelo de maneiras infinitas e torná-lo mais realista. Por exemplo, suponha que um cliente honesto que é fã de um autor de livro sempre dê 5 ao livro, independentemente da qualidade:

Recomendação(c, b) ~ **se** *Honesto* (c) **então**
se *Fã* (c, *Autor* (b)) **então** *Exatamente* (5)
senão *HonestoRecTPC* (*Gentileza* (c), *Qualidade* (b))
senão ⟨0,4, 0,1, 0,0, 0,1,0,4⟩.

Novamente, o teste condicional *Fã*(c, *Autor*(b)) é desconhecido, mas se o cliente dá apenas nota 5 para um autor de livros particular, e não é do tipo especialmente gentil, então é alta a probabilidade posterior que o cliente seja fã do autor. Além disso, a distribuição posterior tenderá a descontar os 5 do cliente na avaliação da qualidade dos livros daquele autor.

No exemplo anterior, assumimos implicitamente que o valor de *Autor*(b) é conhecido para cada b, mas esse pode não ser o caso. Como pode o sistema raciocinar, por exemplo, sobre se C_1 é fã de *Autor*(B_2) quando *Autor*(B_2) é desconhecido? A resposta é que o sistema pode ter de raciocinar sobre todos os autores possíveis. Suponha que (para simplificar) existam apenas dois autores, A_1 e A_2. Então *Autor*(B_2) é uma variável aleatória com dois valores possíveis, A_1 e A_2, e é pai de *Recomendação*(C_1, B_2). As variáveis *Fã*(C_1, A_1) e *Fã* (C_1, A_2) realmente influenciam a recomendação. A Figura 14.19 mostra um fragmento da rede de Bayes equivalente. A incerteza no valor de *Autor*(B_2), que afeta a estrutura de dependência da rede, é um exemplo da **incerteza**

relacional.

No caso de você estar se perguntando como o sistema possivelmente pode descobrir quem é o autor de B_2: considere a possibilidade de que três outros clientes são fãs de A_1 (e não têm outros autores favoritos em comum) e todos três deram 5 a B_2, embora a maioria dos outros clientes ache isso bastante desanimador. Nesse caso, é extremamente provável que A_1 seja o autor de B_2. O surgimento de raciocínio sofisticado como esse de um modelo MPR de apenas algumas linhas é um exemplo interessante de como as influências probabilísticas se espalharam através da rede de interconexões entre os objetos no modelo. Como mais dependências e mais objetos foram adicionados, a imagem transmitida pela distribuição posterior geralmente se torna cada vez mais clara.

A próxima pergunta é como fazer inferência em MPRs. Uma abordagem é coletar as evidências, provas, consultas e símbolos constantes nesse sentido, construir a rede de Bayes equivalente e aplicar qualquer um dos métodos de inferência discutidos neste capítulo. Essa técnica é chamada **desenrolar.** A desvantagem óbvia é que a rede de Bayes resultante pode ser muito grande. Além disso, se houver muitos objetos candidatos de uma relação ou função desconhecida — por exemplo, o autor desconhecido de B_2 —, algumas variáveis na rede podem ter muitos pais.

Felizmente, pode ser feito muito para melhorar os algoritmos de inferência genérica. Primeiro, a presença de estruturas repetidas na rede de Bayes desenrolada significa que muitos dos fatores construídos durante a eliminação de variáveis (e tipos semelhantes de tabelas construídas agrupando algoritmos) será idêntico; esquemas de *caching* eficazes produziram aumentos de velocidade de três ordens de magnitude para grandes redes. Em segundo lugar, os métodos de inferência desenvolvidos para tirar proveito da independência do contexto específico nas redes de Bayes encontram muitas aplicações em MPRs. Terceiro, algoritmos de inferência CMMC têm algumas propriedades interessantes quando aplicados a MPRs com incerteza relacional. O CMMC trabalha com amostragem de mundos completos possíveis, então em cada estado a estrutura relacional é totalmente conhecida. No exemplo dado anteriormente, cada estado CMMC deveria especificar o valor de $Autor(B_2)$ e,assim, os outros autores em potencial não são mais os pais dos nós de recomendação de B_2. Então, a incerteza relacional não causa aumento na complexidade da rede para CMMC; em vez disso, o processo de CMMC inclui transições que alteram a estrutura relacional e, portanto, a estrutura de dependência da rede desenrolada.

Todos os métodos que acabamos de descrever assumem que o MPR tem de ser parcial ou totalmente desenrolado em uma rede bayesiana. Isso é exatamente análogo ao método de **proposicionalização** de inferência lógica de primeira ordem. A resolução de provadores de teorema e de sistemas de programação lógica evita proposicionalizar pela instanciação das variáveis lógicas somente quando necessário para fazer a inferência passar por tudo, isto é, *elevar* o processo de inferência acima do nível de base de sentenças proposicionais e fazer com que cada etapa elevada faça o trabalho de muitas etapas de base. A mesma ideia se aplica à inferência probabilística. Por exemplo, na variável algoritmo de eliminação, um fator elevado pode representar todo um conjunto de fatores de base que atribuem probabilidades a variáveis aleatórias no MPR, onde as variáveis aleatórias diferem apenas nos símbolos constantes usados para construí-las. Os detalhes desse método vão além do escopo deste livro, mas são fornecidas referências ao final do capítulo.

14.6.3 Modelos de probabilidade de universo aberto

Argumentamos anteriormente que a semântica de banco de dados é adequada a situações em que sabemos exatamente o conjunto de objetos relevantes que existem e podem ser identificados de forma inequívoca (em particular, todas as observações sobre um objeto são corretamente associadas ao símbolo constante que o denomina). Em muitos contextos do mundo real, no entanto, esses pressupostos são simplesmente insustentáveis. Demos os exemplos de ISBNs múltiplos e ataques sibila no domínio de recomendação do livro (ao qual retornaremos em um instante), mas o fenômeno é muito mais abrangente:

- Um sistema de visão não sabe o que existe, nem se há algo dobrando a próxima esquina, e pode não saber se o objeto que vê agora é o mesmo que viu há poucos minutos.
- Um sistema de compreensão de texto não sabe com antecedência as entidades que serão apresentadas em um texto e deve inferir sobre se frases como "Mary", "Dr. Smith", "ela", "seu cardiologista", "sua mãe", e assim por diante, referem-se ao mesmo objeto.
- Um analista da inteligência perseguindo espiões nunca sabe realmente quantos espiões existem e só pode adivinhar se vários pseudônimos, números de telefone e avistamentos dizem respeito ao mesmo indivíduo.

Na verdade, grande parte da cognição humana parece exigir saber que objetos existem e, sendo capaz de conectar observações – as quais quase nunca vêm associadas a identificações únicas –, construir hipóteses sobre objetos no mundo.

Por essas razões, precisamos ser capazes de descrever o chamado modelo de probabilidade de **universo aberto** ou MPUOs com base na semântica-padrão de lógica de primeira ordem, como ilustrado na parte superior da Figura 14.18. Uma linguagem de MPUOs fornece uma maneira de escrever esses modelos facilmente enquanto garante uma distribuição de probabilidade única, consistente, sobre o espaço infinito de mundos possíveis.

A ideia básica é entender como as redes bayesianas ordinárias e MPRs definem um modelo de probabilidade único e transferem esse conhecimento para o cenário de primeira ordem. Em essência, uma rede de Bayes *gera* cada mundo possível, evento por evento, na ordem topológica definida pela estrutura de rede, onde cada evento é uma atribuição de um valor a uma variável. Um MPR estende isso para conjuntos inteiros de eventos, definidos pelas instanciações possíveis das variáveis lógicas em determinado predicado ou função. Os MPUOs vão mais longe, permitindo que estapas geradoras *adicionem objetos* ao mundo possível em construção, onde o número e o tipo de objetos podem depender de objetos que já estão nesse mundo. Ou seja, o evento que está sendo gerado não é a atribuição de um valor a uma variável, mas a própria *existência* de objetos.

Uma maneira de fazer isso em MPUOs é adicionar instruções que definem distribuições condicionais sobre os números de objetos de vários tipos. Por exemplo, no domínio de recomendação do livro, podemos querer distinguir entre *clientes* (pessoas reais) e seus *IDs de login*. Suponha que esperamos algo entre 100 e 10.000 clientes distintos (os quais não podemos observar diretamente). Podemos expressar isso como uma distribuição *a priori* log-normal[9] da seguinte forma:

$\# Cliente \sim LogNormal[6,9, 2,3^2]()$.

Esperamos que os clientes honestos tenham apenas um ID, enquanto os clientes desonestos podem ter entre 10 e 1.000 IDs:

$\# LoginID(Proprietário = c) \sim$ **se** $Honesto(c)$, **então** $Exatamente(1)$
senão $LogNormal[6,9, 2,3^2]()$.

Essa declaração define o número de IDs de login de um dado proprietário, que é um cliente. A função *Proprietário* é chamada **função de origem** porque diz onde cada objeto é gerado. Na semântica formal do BLOG (distinta da lógica de primeira ordem), os elementos de domínio em cada mundo possível são realmente histórias de geração (por exemplo, "o quarto ID de login do sétimo cliente"), em vez de símbolos simples.

Sujeitos a condições técnicas de aciclicidade e fundamentação similares aos MPRs, os modelos de universo aberto desse tipo definem uma distribuição única sobre os mundos possíveis. Além disso, existem algoritmos de inferência que, para cada modelo bem definido e para todas as consultas de primeira ordem, retornam resposta que se aproxima da verdade posterior no limite. Há algumas questões complicadas relacionadas ao projeto destes algoritmos. Por exemplo, um algoritmo CMMC não pode fornecer amostra diretamente no espaço de mundos possíveis quando o tamanho desses mundos é ilimitado; em vez disso, ele fornece amostras de mundos finitos, parciais, confiando no fato de que apenas uma quantidade finita de objetos pode ser relevante para a consulta de maneiras distintas. Além disso, as transições devem permitir fundir dois objetos em um ou uma divisão de um objeto em dois (ver detalhes nas referências ao final do capítulo). Apesar dessas complicações, o princípio básico estabelecido na Equação 14.13 ainda é válido: a probabilidade de qualquer sentença é bem definida e pode ser calculada.

As pesquisas nessa área ainda se encontram em fase inicial, mas já está se tornando claro que o raciocínio probabilístico de primeira ordem produz grande aumento na eficácia dos sistemas de IA em lidar com informações incertas. As aplicações potenciais incluem as mencionadas antes — visão computacional, entendimento de texto e análise de informação —, assim como muitos outros tipos de interpretação de sensor.

14.7 OUTRAS ABORDAGENS PARA RACIOCÍNIO INCERTO

Outras ciências (como, por exemplo, física, genética e economia) há muito tempo adotam a probabilidade como um modelo para incerteza. Em 1819, Pierre Laplace disse: "A teoria da probabilidade não é nada além do bom senso reduzido ao cálculo." Em 1850, James Maxwell afirmou: "A verdadeira lógica para esse mundo é o cálculo de probabilidades, que leva em conta a magnitude da probabilidade que está ou deveria estar na mente de um homem razoável."

Dada essa longa tradição, talvez seja surpreendente que a IA tenha considerado muitas alternativas para a probabilidade. Os primeiros sistemas especialistas da década de 1970 ignoravam a incerteza e usavam o raciocínio lógico estrito, mas logo ficou claro que isso era impraticável para a maioria dos domínios do mundo real. A geração seguinte de sistemas especialistas (especialmente em

domínios médicos) empregava técnicas probabilísticas. Os resultados iniciais foram promissores, mas não puderam ser usados em larga escala devido ao número exponencial de probabilidades exigidas na distribuição conjunta total (os algoritmos eficientes de redes bayesianas eram desconhecidos na época). Como resultado, as abordagens probabilísticas perderam a preferência no período entre 1975 e 1988, e foram experimentadas diversas alternativas para probabilidade, por várias razões:

- Uma visão comum é que a teoria da probabilidade é essencialmente numérica, enquanto o raciocínio de bom senso do ser humano é mais "qualitativo". Certamente, não temos consciência da necessidade de efetuar cálculos numéricos de graus de crença (nem estamos cientes da necessidade de efetuar a unificação, ainda que aparentemente sejamos capazes de efetuar raciocínio lógico). É possível que tenhamos algum tipo de graus numéricos de crença codificados diretamente em forças de conexões e ativações em nossos neurônios. Nesse caso, a dificuldade de acesso consciente a essas forças não surpreende. Também devemos observar que os mecanismos de raciocínio qualitativo podem ser construídos diretamente sobre a teoria da probabilidade, de forma que o argumento contrário à probabilidade baseado na ideia de "nada de números" tem pouca força. Todavia, alguns esquemas qualitativos têm bastante apelo por si só. Um dos mais bem estudados é o **raciocínio default**, que trata as conclusões não como "dignas de confiança até certo grau", mas como "dignas de confiança até se encontrar uma razão melhor para acreditar em algo diferente". O raciocínio default é estudado no Capítulo 12.
- As abordagens **baseadas em regras** para a incerteza também foram experimentadas. Tais abordagens esperam usar como fundamento o sucesso de sistemas baseados em regras lógicas, mas acrescentam uma espécie de "fator de confusão" a cada regra para acomodar a incerteza. Esses métodos foram desenvolvidos em meados da década de 1970 e formaram a base para um grande número de sistemas especialistas em medicina e outras áreas.
- Uma área que não estudamos até agora é a questão da **ignorância**, em oposição à incerteza. Considere o lançamento de uma moeda. Se soubermos que a moeda é imparcial, é razoável admitir a probabilidade de 0,5 para cara. Se soubermos que a moeda é parcial, mas não soubermos como, então 0,5 será a única probabilidade razoável. É óbvio que os dois casos são diferentes, ainda que o resultado da probabilidade pareça não fazer distinção entre eles. A **teoria de Dempster-Shafer** utiliza graus de crença com **valores de intervalos** para representar o conhecimento de um agente sobre a probabilidade de uma proposição.
- A probabilidade assume o mesmo compromisso ontológico da lógica: que as proposições sejam verdadeiras ou falsas no mundo, ainda que o agente esteja inseguro sobre qual seja o caso. Os pesquisadores em **lógica difusa** propuseram uma ontologia que permite a **imprecisão**: uma proposição pode ter "um grau de" verdade. A imprecisão e a incerteza são de fato questões ortogonais, como veremos.

As três subseções a seguir tratam de algumas dessas abordagens com um pouco mais de profundidade. Não forneceremos material técnico detalhado, mas citaremos referências para estudo adicional.

14.7.1 Métodos baseados em regras para raciocínio incerto

Os sistemas baseados em regras emergiram do trabalho inicial sobre sistemas práticos e intuitivos para inferência lógica. Os sistemas lógicos em geral e os sistemas baseados em regras lógicas em particular têm três propriedades interessantes:

- **Localidade:** Em sistemas lógicos, sempre que temos uma regra da forma $A \Rightarrow B$, podemos concluir B, dada a evidência A, *sem preocupação com quaisquer outras regras*. Em sistemas probabilísticos, precisamos considerar *toda* evidência.
- **Separação:** Uma vez que uma prova lógica seja encontrada para uma proposição B, a proposição pode ser usada, independentemente do modo como foi derivada. Ou seja, ela pode ser **separada** de sua justificativa.Por outro lado, ao lidar com probabilidades, a origem da evidência de uma crença é importante para o raciocínio subsequente.
- **Verdade-funcionalidade:** Em lógica, a verdade de sentenças complexas pode ser calculada a partir da verdade dos componentes. A combinação de probabilidades não funciona desse modo, exceto sob suposições fortes de independência global.

Houve várias tentativas para criar esquemas de raciocínio incerto que retêm essas vantagens. A ideia é associar graus de crença a proposições e regras, e gerar esquemas puramente locais para combinar e propagar esses graus de crença. Os esquemas também são verdade-funcionais; por exemplo, o grau de crença em $A \lor B$ é uma função da crença em A e da crença em B. A má notícia para sistemas baseados em regras é que as propriedades de *localidade*, *separação e verdade-funcionalidade simplesmente não são apropriadas para o raciocínio incerto*. Vamos examinar primeiro a verdade-funcionalidade. Seja H_1 o evento em que um lançamento de moeda imparcial resulta em cara, seja T_1 o evento em que a moeda resulta em coroa nesse mesmo lançamento e seja H_2 o evento em que a moeda mostra cara em um segundo lançamento. É claro que os três eventos têm a mesma probabilidade, 0,5, e portanto um sistema verdade-funcional deve atribuir a mesma crença à disjunção de dois desses eventos. No entanto, podemos ver que a probabilidade da disjunção depende dos próprios eventos e não apenas de suas probabilidades:

$P(A)$	$P(B)$	$P(A \lor B)$
$P(H_1) = 0{,}5$	$P(H_1) = 0{,}5$	$P(H_1 \lor H_1) = 0{,}50$
	$P(T_1) = 0{,}5$	$P(H_1 \lor T_1) = 1{,}00$
	$P(H_2) = 0{,}5$	$P(H_1 \lor H_2) = 0{,}75$

A situação fica pior quando juntamos a evidência. Os sistemas verdade-funcionais têm **regras** da forma $A \mapsto B$ que nos permitem calcular a crença em B como uma função da crença na regra e a crença em A. Podem ser criados sistemas de encadeamento para a frente e de encadeamento para trás. A crença na regra é considerada constante e, em geral, é especificada pelo engenheiro do conhecimento — por exemplo, como $A \mapsto_{0,9} B$.

Considere a situação de grama molhada da Figura 14.12(a). Se quiséssemos ter a possibilidade de

desenvolver tanto o raciocínio causal quanto o raciocínio de diagnóstico, precisaríamos das duas regras a seguir:

$$Chuva \mapsto GramaMolhada \qquad \text{e} \qquad GramaMolhada \mapsto Chuva\,.$$

Essas duas regras formam um laço de realimentação: a evidência de *Chuva* aumenta a crença em *GramaMolhada* que, por sua vez, aumenta ainda mais a crença em *Chuva*. É claro que os sistemas de raciocínio incerto precisam manter o controle dos caminhos ao longo dos quais a evidência é propagada.

O raciocínio intercausal (ou de explicação muito boa) também é complicado. Considere o que acontece quando temos as duas regras:

$$Irrigador \mapsto GramaMolhada \qquad \text{e} \qquad GramaMolhada \mapsto Chuva.$$

Vamos supor que observamos que o irrigador está ligado. Por encadeamento direto através de nossas regras, isso aumenta a crença de que a grama ficará molhada, o que, por sua vez, aumenta a crença de que está chovendo. Entretanto, isso é ridículo: o fato de o irrigador estar funcionando explica muito bem a grama molhada e deve *reduzir* a crença de chuva. Um sistema verdade-funcional atua como se também acreditasse que $Irrigador \mapsto Chuva$.

Dadas essas dificuldades, como é possível que sistemas verdade-funcionais podem ser úteis na prática? A resposta reside na restrição da tarefa e em cuidadosa engenharia da base de regras para que interações indesejáveis não ocorram. O exemplo mais famoso de um sistema verdade-funcional para raciocínio incerto é o modelo de **fatores de certeza**, desenvolvido para o programa de diagnóstico médico MYCIN e que foi amplamente utilizado em sistemas especialistas do final da década de 1970 e na década de 1980. Quase todos os usos de fatores de certeza envolviam conjuntos de regras que eram puramente de diagnóstico (como no MYCIN) ou puramente causais. Além disso, a evidência só foi introduzida nas "raízes" do conjunto de regras, e a maioria dos conjuntos de regras era unicamente conectada. Heckerman (1986) mostrou que, sob essas circunstâncias, uma variação secundária na inferência do fator de certeza era exatamente equivalente à inferência bayesiana em poliárvores. Em outras circunstâncias, os fatores de certeza poderiam resultar em graus de crença desastrosamente incorretos, devido à superestimativa da evidência. À medida que os conjuntos de regras ficavam maiores, interações indesejáveis entre as regras se tornavam mais comuns, e os médicos descobriram que os fatores de certeza de muitas outras regras tinham de ser "adaptados" quando novas regras eram adicionadas. Por essas razões, as redes bayesianas suplantaram os métodos baseados em regra de raciocínio incerto.

14.7.2 Representação da ignorância: teoria de Dempster-Shafer

A **teoria de Dempster-Shafer** foi criada para lidar com a distinção entre **incerteza** e **ignorância**. Em vez de calcular a probabilidade de uma proposição, ela calcula a probabilidade de que a evidência admita a proposição. Essa medida de crença é chamada **função de crença**, representada por $Bel(X)$.

Voltamos ao lançamento de moedas para ver um exemplo de funções de crença. Suponha que você

pegue uma moeda do bolso de um mágico. Dado que a moeda pode ou não ser imparcial, que crença você deve atribuir ao evento de que o lançamento dará cara? A teoria de Dempster-Shafer lhe diz que, como não há nenhuma evidência em contrário, você tem de afirmar que a crença $Bel(Cara) = 0$ e também que $Bel(\neg Cara) = 0$. Isso torna os sistemas de raciocínio de Dempster-Shafer céticos de um modo que apresenta certo apelo intuitivo. Agora, suponha que você tenha um especialista à sua disposição que atesta com 90% de certeza que a moeda é imparcial (isto é, ele está 90% certo de que $P(Cara) = 0{,}5$). Então, a teoria de Dempster-Shafer resulta em $Bel(Cara) = 0{,}9 \times 0{,}5 = 0{,}45$ e, da mesma forma, em $Bel(\neg Cara) = 0{,}45$. Existe ainda um "intervalo" de 10 pontos percentuais não contabilizado pela evidência.

Os fundamentos matemáticos da teoria de Dempster-Shafer têm um sabor semelhante ao da teoria da probabilidade; a principal diferença é que, em vez de atribuir probabilidades para mundos possíveis, a teoria atribui **massas** para conjuntos de mundos possíveis, isto é, eventos. As massas ainda devem adicionar 1 sobre todos os eventos possíveis. $Bel(A)$ é definido como sendo a soma das massas para todos os eventos que são subconjuntos de A (ou seja, que implicam em A), incluindo o próprio A. Com essa definição, a soma de $Bel(A)$ e $Bel(\neg A)$ é no *máximo* 1, e o intervalo entre $Bel(A)$ e $1 - Bel(\neg A)$ é muitas vezes interpretado como limitando a probabilidade de A.

Assim como ocorre com o raciocínio default, há um problema na conexão entre crenças e ações. Sempre que existe uma lacuna nas crenças, um problema de decisão pode ser definido de tal forma que o sistema de Dempster-Shafer é incapaz de tomar uma decisão. Na verdade, a noção de utilidade no modelo de Dempster-Shafer ainda não está bem entendida, pois os significados das massas e crenças em si ainda não foram bem compreendidos. Pearl (1988) argumentou que $Bel(A)$ deve ser interpretado não como um grau de crença em A, mas como a probabilidade atribuída a todos os mundos possíveis (agora interpretados como teorias lógicas) em que A é *demonstrável*. Embora existam casos em que essa quantidade possa ser de interesse, ela não é o mesmo que a probabilidade de que A seja verdadeiro.

Uma análise bayesiana do lançamento da moeda sugere que não é necessário nenhum formalismo novo para lidar com esses casos. O modelo teria duas variáveis: a *Parcialidade* da moeda (um número entre 0 e 1, onde 0 é uma moeda que sempre mostra coroa e 1 é uma moeda que sempre mostra cara) e o resultado do próximo *Lance*. A distribuição de probabilidade *a priori* para *Parcialidade* refletiria nossas crenças com base na origem da moeda (o bolso do mágico): uma pequena probabilidade que seja imparcial e outra que seja fortemente inclinada a dar cara ou coroa. A distribuição condicional $P(Lance \mid Parcial)$ simplesmente define como opera a parcialidade. Se $P(Parcial)$ for simétrico em torno de 0,5, então a nossa probabilidade prévia para o lance é de

$$P(Lance = cara) = \int_0^1 P(Parcial = x)\, P(Lance = cara \mid Parcial = x)\, dx = 0{,}5.$$

Essa é a mesma previsão que teríamos se acreditássemos fortemente que a moeda fosse parcial, mas *não* significa que a teoria da probabilidade trata as duas situações de forma idêntica. A diferença surge *após* o lance, ao calcular a distribuição posterior da *Parcialidade*. Se a moeda veio de um banco, então verificar que deu cara três vezes quase não abalaria a nossa forte crença anterior na sua imparcialidade; mas, se a moeda vier do bolso do mágico, a mesma evidência levará a uma crença posterior mais forte de que a moeda tende a dar coroa. Assim, uma abordagem bayesiana

expressa nossa "ignorância" em termos de como nossas crenças mudariam em face da coleta de informações futuras.

14.7.3 Representação da imprecisão: conjuntos difusos e lógica difusa

☞ A **teoria de conjuntos difusos** é um meio de especificar o quanto um objeto satisfaz uma descrição vaga. Por exemplo, considere a proposição: "Nei é alto." Isso é verdadeiro se Nei tiver 1,78 m? A maioria das pessoas hesitaria em responder "verdadeiro" ou "falso", preferindo dizer: "Talvez." Observe que isso não é uma questão de incerteza sobre o mundo exterior — estamos certos da altura de Nei. A questão é que o termo linguístico "alto" não se refere a uma demarcação nítida de objetos em duas classes — existem *graus* de altura. Por essa razão, *a teoria de conjuntos difusos não é de forma alguma um método para raciocínio incerto*. Em vez disso, a teoria de conjuntos difusos trata *Alto* como um predicado difuso e afirma que o valor-verdade de *Alto*(*Nei*) é um número entre 0 e 1, em vez de ser simplesmente *verdadeiro* ou *falso*. O nome "conjuntos difusos" deriva da interpretação do predicado como a definição implícita de um conjunto de seus elementos — um conjunto que não tem limites precisos.

A **lógica difusa** é um método para raciocínio com expressões lógicas que descrevem a pertinência a conjuntos difusos. Por exemplo, a sentença complexa *Alto*(*Nei*) ∧ *Pesado*(*Nei*) tem um valor-verdade difuso que é uma função dos valores-verdade de seus componentes. As regras-padrão para avaliar a verdade difusa, *V*, de uma sentença complexa são:

$$V(A \land B) = \min(V(A), V(B))$$
$$V(A \lor B) = \max(V(A), V(B))$$
$$V(\neg A) = 1 - V(A).$$

A lógica difusa é um sistema verdade-funcional — um fato que causa sérias dificuldades. Por exemplo, suponha que $V(Alto(Nei)) = 0{,}6$ e $V(Pesado(Nei)) = 0{,}4$. Portanto, temos $V(Alto(Nei) \land Pesado(Nei)) = 0{,}4$, que parece razoável,mas também obtemos o resultado $V(Alto(Nei) \land \neg Alto(Nei)) = 0{,}4$, que não parece nem um pouco razoável. Sem dúvida, o problema surge da inabilidade de uma abordagem verdade-funcional levar em conta as correlações ou anticorrelações entre as proposições componentes.

O **controle difuso** é uma metodologia para construir sistemas de controle em que o mapeamento entre os parâmetros de entrada e saída com valores reais é representado por regras difusas. O controle difuso tem sido muito bem-sucedido em produtos comerciais como transmissões automáticas, câmeras de vídeo e barbeadores elétricos. Os críticos (veja, por exemplo, Elkan, 1993) argumentam que essas aplicações são bem-sucedidas porque têm bases de regras pequenas, nenhum encadeamento de inferências e parâmetros harmoniosos que podem ser ajustados para melhorar o desempenho do sistema. O fato de eles serem implementados com operadores difusos poderia ser apenas circunstancial; a chave é simplesmente fornecer um modo conciso e intuitivo de especificar uma função de valores reais suavemente interpolada. Houve tentativas para fornecer uma explicação da lógica difusa em termos da teoria da probabilidade. Uma ideia é visualizar asserções do tipo "Nei é alto" como observações discretas feitas em relação a uma variável contínua oculta, a *Altura* real de

Nei. O modelo de probabilidade especifica *P*(O observador afirma que Nei é alto | *Altura*), talvez usando uma **distribuição probit**, como descrito anteriormente. Uma distribuição posterior sobre a altura de Nei pode então ser calculada da maneira usual, por exemplo, se o modelo fizer parte de uma rede bayesiana híbrida. É claro que tal abordagem não é verdade-funcional. Por exemplo, a distribuição condicional:

P(O observador afirma que Nei é alto e pesado | *Altura*, *Peso*)

permite interações entre altura e peso para formar a causa da observação. Desse modo, é muito improvável que alguém que tenha 2,43 metros de altura e pese 86 quilos seja chamado de “alto e pesado”, embora “2,43 metros” signifique “alto” e “86 quilos” signifique “pesado”.

Os predicados difusos também podem receber uma interpretação probabilística em termos de **conjuntos aleatórios**, isto é, variáveis aleatórias cujos valores possíveis são conjuntos de objetos. Por exemplo, *Alto* é um conjunto aleatório cujos valores possíveis são conjuntos de pessoas. A probabilidade $P(Alto = S_1)$, onde S_1 é algum conjunto específico de pessoas, é a probabilidade de que exatamente esse conjunto seja identificado como “alto” por um observador. Então, a probabilidade de “Nei é alto” é a soma das probabilidades de todos os conjuntos dos quais Nei é um elemento.

Tanto a abordagem de rede bayesiana híbrida quanto a abordagem de conjuntos aleatórios parecem captar aspectos de imprecisão sem introduzir graus de verdade. Todavia, permanecem abertas muitas questões relativas à representação apropriada de observações linguísticas e quantidades contínuas — questões que foram negligenciadas pela maioria dos pesquisadores de fora da comunidade difusa.

14.8 RESUMO

Este capítulo descreveu as **redes bayesianas**, uma representação bem desenvolvida para o conhecimento incerto. As redes bayesianas desempenham um papel aproximadamente análogo ao da lógica proposicional para o conhecimento definido.

- Uma rede bayesiana é um grafo acíclico orientado cujos nós correspondem a variáveis aleatórias; cada nó tem uma distribuição condicional para o nó, dados seus pais.
- As redes bayesianas fornecem um modo conciso de representar relacionamentos de **independência condicional** no domínio.
- Uma rede bayesiana especifica uma distribuição conjunta total; cada entrada conjunta é definida como o produto das entradas correspondentes nas distribuições condicionais locais. Uma rede bayesiana com frequência é exponencialmente menor que a distribuição conjunta enumerada explicitamente.
- Muitas distribuições condicionais podem ser representadas de forma compacta por famílias canônicas de distribuições. As **redes bayesianas híbridas**, que incluem tanto variáveis discretas quanto variáveis contínuas, utilizam uma variedade de distribuições canônicas.
- A inferência em redes bayesianas significa calcular a distribuição de probabilidade de um conjunto de variáveis de consulta, dado um conjunto de variáveis de evidência. Os algoritmos de

inferência exata, como a **eliminação de variáveis**, avaliam somas de produtos de probabilidades condicionais de forma tão eficiente quanto possível.

- Em **poliárvores** (redes unicamente conectadas), a inferência exata demora um tempo linear no tamanho da rede. No caso geral, o problema é intratável.
- Técnicas de aproximação estocástica como **ponderação de probabilidade** e **cadeia de Markov Monte Carlo** podem fornecer estimativas razoáveis das probabilidades posteriores verdadeiras em uma rede e lidar com redes muito maiores do que os algoritmos exatos.
- A teoria da probabilidade pode ser combinada com ideias de representação da lógica de primeira ordem para produzir sistemas muito poderosos destinados ao raciocínio sob incerteza. O s **modelos de probabilidade relacional** (MPRs) incluem restrições de representação que garantem uma distribuição de probabilidade bem definida que pode ser expressa como uma rede bayesiana equivalente. Os **modelos de probabilidade de universo aberto** tratam da existência e da **incerteza da identidade**, definindo distribuições de probabilidade sobre o espaço infinito dos mundos possíveis de primeira ordem.
- Vários sistemas alternativos para raciocínio sob incerteza foram sugeridos. De modo geral, sistemas **verdade-funcionais** não são adequados para tal raciocínio.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O uso de redes para representar informações probabilísticas começou no início do século XX, com o trabalho de Sewall Wright sobre a análise probabilística de herança genética e fatores de crescimento dos animais (Wright, 1921, 1934). I. J. Good (1961), em colaboração com Alan Turing, desenvolveu representações probabilísticas e métodos de inferência de Bayes que poderiam ser considerados precursores das redes bayesianas modernas, embora o artigo não seja citado com frequência nesse contexto.[10] O mesmo artigo é a fonte original do modelo de OU ruidoso.

A representação de **diagrama de influência** para problemas de decisão, que incorporou uma representação de GAO para variáveis aleatórias, foi usada em análise de decisão no final dos anos 1970 (veja o Capítulo 16), mas apenas a enumeração foi usada para avaliação. Judea Pearl desenvolveu o método de passagem de mensagens para transporte de inferência em redes de árvores (Pearl, 1982a) e em redes de poliárvores (Kim e Pearl, 1983) e explicou a importância de se construírem modelos de probabilidade causal em lugar de modelos de probabilidade de diagnóstico, em contraste com os sistemas de fatores de certeza então em voga.

O primeiro sistema especialista a utilizar redes bayesianas foi o CONVINCE (Kim, 1983). Aplicações antigas em medicina incluem o sistema MUNIN para diagnosticar enfermidades neuromusculares (Andersen *et al*., 1989) e o sistema PATHFINDER para patologia (Heckerman, 1991). O sistema CPCS (Pradhan *et al*., 1994) é uma rede bayesiana para a medicina interna consistindo de 448 nós, 906 ligações e 8.254 valores de probabilidade condicional (a capa mostra uma parte da rede).

As aplicações em engenharia incluem o trabalho do Electric Power Research Institute sobre monitoramento de geradores de energia (Morjaria *et al*., 1995), o trabalho da Nasa sobre a exibição de informação de tempo crítico no controle de missão em Houston (Horvitz e Barry, 1995), e o

campo geral da **tomografia de rede**, que visa inferir propriedades locais de nós despercebidos e links na Internet a partir de observações sobre o desempenho de mensagem de ponta a ponta (Castro *et al*., 2004). Talvez a rede bayesiana utilizada mais amplamente tenha sido os módulos de diagnose e reparação (por exemplo, o Printer Wizard) no Microsoft Windows (Breese e Heckerman, 1996) e o Office Assistant da Microsoft (Horvitz *et al*., 1998). Outra área de aplicação importante é a biologia: têm sido utilizadas redes bayesianas para a identificação de genes humanos por referência a genes de ratos (Zhang *et al*., 2003), inferindo redes celulares Friedman (2004) e muitas outras tarefas em bioinformática. Poderíamos continuar, mas, em vez disso, vou encaminhá-lo para Pourret *et al.* (2008), um guia de 400 páginas para aplicações de redes bayesianas.

Ross Shachter (1986), trabalhando na comunidade do diagrama de influência, desenvolveu o primeiro algoritmo completo para redes bayesianas gerais. Seu método era baseado em redução orientada ao objetivo da rede, utilizando transformações de preservação posterior. Pearl (1986) desenvolveu um algoritmo de agrupamento para inferência exata em redes bayesianas gerais, utilizando uma conversão para uma poliárvore de agrupamentos orientada, em que a passagem de mensagens foi utilizada para obter consistência sobre as variáveis compartilhadas entre agrupamentos. Uma abordagem semelhante, desenvolvida pelos estatísticos David Spiegelhalter e Steffen Lauritzen (Lauritzen e Spiegelhalter, 1988), baseia-se em conversão para uma forma não orientada do modelo gráfico chamado **rede de Markov**. Essa abordagem foi implementada no sistema Hugin, uma ferramenta eficaz e amplamente utilizada para raciocínio incerto (Andersen *et al*., 1989). Boutilier *et al*. (1996) mostram como explorar a independência de contexto específico em algoritmos de agrupamento.

A ideia básica de eliminação de variável — de que cálculos repetidos dentro da expressão geral de soma de produtos podem ser evitados por *caching* — apareceu no algoritmo de inferência probabilística simbólica (SPI) (Shachter *et al.,* 1990). O algoritmo de eliminação que descrevemos está mais próximo daquele que foi desenvolvido por Zhang e Poole (1994). Critérios para podar variáveis irrelevantes foram desenvolvidos por Geiger *et al.* (1990) e por Lauritzen *et al.* (1990); o critério que apresentamos é um caso especial simples desses critérios. Dechter (1999) mostra como a ideia de eliminação de variáveis é essencialmente idêntica à **programação dinâmica não serial** (Bertele e Brioschi, 1972), uma abordagem algorítmica que pode ser aplicada para resolver uma variedade de problemas de inferência em redes bayesianas — como, por exemplo, encontrar a **explicação mais provável** para um conjunto de observações. Esse método conecta algoritmos de redes bayesianas inter-relacionados para resolver PSRs e apresenta uma medida direta da complexidade da inferência exata em termos da largura de árvore da rede. Wexler e Meek (2009) descreveram um método de impedir o crescimento exponencial do tamanho dos fatores calculados na eliminação de variáveis; seu algoritmo divide os fatores grandes em produtos de fatores menores e calcula simultaneamente um limite de erro de aproximação resultante.

A inclusão de variáveis aleatórias contínuas em redes bayesianas foi considerada por Pearl (1988) e por Shachter e Kenley (1989); esses documentos discutiam redes contendo apenas variáveis contínuas com distribuições gaussianas lineares. A inclusão de variáveis discretas foi investigada por Lauritzen e Wermuth (1989) e implementada no sistema cHUGIN (Olesen, 1993). A análise mais aprofundada dos modelos lineares de Gauss, com conexões para muitos outros modelos utilizados nas estatísticas, aparece em Roweis e Ghahramani (1999). A distribuição probit é geralmente

atribuída a Gaddum (1933) e Bliss (1934), embora tivesse sido descoberta várias vezes no século XIX. O trabalho de Bliss foi expandido consideravelmente por Finney (1947). O probit foi amplamente utilizado para modelar fenômenos de escolha discreta e pode ser estendido para lidar com mais de duas escolhas (Daganzo, 1979). O modelo logit foi introduzido por Berkson (1944); inicialmente muito ridicularizado, ele se tornou mais popular que o modelo probit. Bishop (1995) oferece uma justificação simples para o seu uso.

Cooper (1990) mostrou que o problema geral de inferência em redes bayesianas irrestritas é NP-difícil, e Paul Dagum e Mike Luby (1993) mostraram que o problema de aproximação correspondente é NP-difícil. A complexidade de espaço também é um problema sério em métodos de formação de agrupamentos e de eliminação de variáveis. O método de **condicionamento de conjunto de corte**, desenvolvido para PSRs no Capítulo 6, evita a construção de tabelas exponencialmente grandes. Em uma rede bayesiana, um conjunto de corte é um conjunto de nós que, quando instanciado, reduz os nós restantes a uma poliárvore que pode ser resolvida em tempo e espaço lineares. A consulta é respondida efetuando-se o somatório de todas as instanciações do conjunto de corte e, portanto, o requisito de espaço global ainda é linear (Pearl, 1988). Darwiche (2001) descreve um algoritmo de condicionamento recursivo que permite um intervalo completo de compromissos de espaço/tempo.

O desenvolvimento de algoritmos de aproximação rápida para inferência de redes bayesianas é uma área muito ativa, com contribuições da estatística, da ciência da computação e da física. O método de amostragem de rejeição é uma técnica geral conhecida há muito tempo pelos estatísticos; ela foi aplicada primeiro a redes bayesianas por Max Henrion (1988), que a denominou **amostragem de lógica**. A ponderação de probabilidade, desenvolvida por Fung e Chang (1989) e por Shachter e Peot (1989), é um exemplo do método estatístico bem conhecido de **amostragem de importância**. Cheng e Druzdzel (2000) descrevem uma versão adaptável da ponderação de probabilidade que funciona bem até mesmo quando a evidência tem uma probabilidade *a priori* muito baixa.

Os algoritmos de cadeia de Markov Monte Carlo (CMMC) começaram com o algoritmo de Metropolis, devido a Metropolis *et al.* (1953), que também foi a origem do algoritmo de têmpera simulada descrito no Capítulo 4. O sistema de amostragem de Gibbs foi criado por Geman e Geman (1984) para inferência em redes de Markov não orientadas. A aplicação de CMMC a redes bayesianas se deve a Pearl (1987). Os documentos reunidos por Gilks *et al.* (1996) abrangem ampla variedade de aplicações de CMMC, várias das quais foram desenvolvidas no bem conhecido pacote BUGS (Gilks *et al.*, 1994).

Existem duas famílias muito importantes de métodos de aproximação que não cobrimos no capítulo. A primeira é a família de métodos de **aproximação variacional**, que podem ser usados para simplificar cálculos complexos de todos os tipos. A ideia básica é propor uma versão reduzida do problema original que seja simples de utilizar, mas que lembre tão aproximadamente quanto possível o problema original. O problema reduzido é descrito por alguns **parâmetros variacionais** λ que são ajustados para minimizar uma função de distância D entre o problema original e o problema reduzido, frequentemente pela resolução do sistema de equações $\partial D/\partial \lambda = 0$. Em muitos casos, podem ser obtidos limites rígidos superiores e inferiores. Os métodos variacionais são empregados há longo tempo em estatística (Rustagi, 1976). Em física estatística, o método de **campo médio** é uma aproximação variacional específica em que se supõe que as variáveis individuais que constituem o

modelo são completamente independentes. Essa ideia foi aplicada à resolução de grandes redes de Markov não orientadas (Peterson e Anderson, 1987; Parisi, 1988). Saul *et al.* (1996) desenvolveram os fundamentos matemáticos para a aplicação de métodos variacionais a redes bayesianas e obtiveram aproximações com limites inferiores mais precisos para redes de sigmoides com o emprego de métodos de campo médio. Jaakkola e Jordan (1996) estenderam a metodologia para obter tanto limites inferiores quanto superiores. Desde esses primeiros trabalhos, métodos variacionais foram aplicados a muitas famílias de modelos específicos. O artigo notável de Wainwright e Jordan (2008) fornece uma unificação da análise teórica da literatura sobre métodos variacionais.

Uma segunda família importante de algoritmos de aproximação se baseia no algoritmo de passagem de mensagens de poliárvore de Pearl (1982a). Esse algoritmo pode ser aplicado a redes gerais, conforme foi sugerido por Pearl (1988). Os resultados podem ser incorretos ou o algoritmo pode deixar de terminar, mas, em muitos casos, os valores obtidos estão próximos dos valores verdadeiros. Pouca atenção foi dada a essa abordagem, denominada **propagação de crença** (ou PC), até McEliece *et al.* (1998) observarem que a passagem de mensagens em uma rede bayesiana com várias conexões era exatamente a computação executada pelo algoritmo de **decodificação turbo** (Berrou *et al.*, 1993), o que proporcionou uma inovação importante no projeto de códigos eficientes de correção de erros. A implicação é que PC é ao mesmo tempo rápida e precisa nas redes muito grandes e altamente conectadas usadas para decodificação, e poderia portanto ser útil em aplicações mais gerais. Murphy *et al.* (1999) apresentaram um estudo empírico promissor do desempenho do PC, e Weiss e Freeman (2001) estabeleceram fortes resultados de convergência para o PC em redes lineares de Gauss. Weiss (2000b) mostrou como funciona uma aproximação chamada propagação de crença por laço e quando a aproximação está correta. Yedidia *et al.* (2001) descobriram conexões adicionais entre a propagação com laços e ideias da física estatística.

A conexão entre probabilidade e linguagens de primeira ordem foi estudada inicialmente por Carnap (1950). Gaifman (1964), bem como Scott e Krauss (1966), definiram uma linguagem em que as probabilidades podiam ser associadas a sentenças de primeira ordem e para a qual os modelos eram medidas de probabilidade sobre mundos possíveis. Dentro da IA, essa ideia foi desenvolvida para a lógica proposicional por Nilsson (1986) e para a lógica de primeira ordem por Halpern (1990). A primeira investigação extensa de questões de representação de conhecimento em tais linguagens foi realizada por Bacchus (1990). A ideia básica era que cada sentença na base de conhecimento expresse uma *restrição* na distribuição sobre mundos possíveis; uma sentença permite que a outra expresse uma restrição forte. Por exemplo, a sentença $\forall x\, P(Faminto\ (x)) > 0{,}2$ exclui distribuições em que qualquer objeto esteja faminto, com probabilidade inferior a 0,2; portanto, permite a sentença $\forall x\, P(Faminto\ (x)) > 0{,}1$. Acontece que escrever um conjunto *consistente* de sentenças nessas linguagens é bastante difícil e construir um modelo único de probabilidade é quase impossível, a menos que se adote a abordagem de representação de redes bayesianas escrevendo sentenças adequadas sobre probabilidades condicionais.

No início dos anos 1990, pesquisadores trabalhando em aplicações complexas, observaram as limitações expressivas das redes bayesianas e desenvolveram várias linguagens para escrever “modelos” com variáveis lógicas, a partir das quais se pode construir grandes redes automaticamente para cada instância do problema (Breese, 1992; Wellman *et al.*, 1992). A linguagem mais importante

foi BUGS (inferência bayesiana usando amostragem de Gibbs) (Gilks *et al*., 1994), que combinou redes bayesianas com a notação da **variável aleatória indexada** comum em estatística (em BUGS, uma variável indexada aleatória se parece com $X[i]$, onde i tem um intervalo de números inteiros definido). Essas linguagens herdaram a propriedade fundamental das redes bayesianas: toda a base de conhecimento bem formada define um modelo de probabilidade consistente e único. Linguagens com semânticas bem definidas com base em nomes únicos e domínio fechado ocasionam capacidades representacionais da programação lógica (Poole, 1993; Sato e Kameya, 1997; Kersting *et al*., 2000) e de redes semânticas (Koller e Pfeffer, 1998; Pfeffer, 2000). Pfeffer (2007) continuou a desenvolver o IBAL, que representa modelos de probabilidade de primeira ordem como programas probabilísticos em uma linguagem de programação estendida com randomização primitiva. Outro segmento importante foi a combinação de notações relacional e de primeira ordem com as redes (não orientadas) de Markov (Taskar *et al*., 2002; Domingos e Richardson, 2004), onde a ênfase está menos na representação do conhecimento e mais em aprender com grandes conjuntos de dados.

Inicialmente, a inferência nesses modelos foi realizada através da geração de uma rede bayesiana equivalente. Pfeffer *et al*. (1999) introduziram um algoritmo de eliminação de variáveis que colocava em cache cada fator calculado para reutilização por cálculos posteriores envolvendo as mesmas relações mas objetos diferentes, tornando real então parte dos ganhos computacionais da elevação. O primeiro algoritmo de inferência realmente elevado foi uma forma elevada de eliminação de variável descrita por Poole (2003) e posteriormente melhorado por Salvo Braz *et al*. (2007). Outros avanços, incluindo casos em que certas probabilidades agregadas podem ser calculadas de forma fechada, são descritos por Milch *et al*. (2008) e Kisynski e Poole (2009). Pasula e Russell (2001) estudaram a aplicação CMMC para evitar a construção da rede de Bayes equivalente completa em casos de incerteza relacional e de identidade. Getoor e Taskar (2007) reuniram muitas teses importantes sobre modelos e probabilidades de primeira ordem e seu uso na aprendizagem de máquina.

O raciocínio probabilístico sobre a incerteza de identidade tem duas origens distintas. Em estatística, surge o problema de **ligação de registro** quando os registros de dados não contêm identificadores únicos padrão, por exemplo, diversas citações deste livro podem denominar seu primeiro autor “Stuart Russell”ou “S. J. Russell” ou mesmo “Stewart Russle”, e outros autores podem usar alguns dos mesmos nomes. Literalmente centenas de empresas existem apenas para resolver problemas de ligação de registros financeiros, médicos, censos e outros dados. A análise probabilística volta ao trabalho de Dunn (1946); o modelo Fellegi-Sunter (1969), que é essencialmente Bayes ingênuo aplicado à combinação, ainda domina a prática atual. A segunda origem para o trabalho sobre a incerteza de identidade é de rastreamento multialvo (Sittler, 1964), que cobriremos no Capítulo 15. Para a maioria de sua história, trabalhar com IA simbólico assumiu erroneamente que os sensores poderiam fornecer sentenças com identificadores únicos de objetos. A questão foi estudada no contexto de compreensão da linguagem por Charniak e Goldman (1992) e no contexto da vigilância por Huang e Russell (1998) e Pasula *et al.* (1999). Pasula *et al.* (2003) desenvolveram um modelo complexo gerador para autores, teses e cadeias de citação, envolvendo tanto incerteza relacional como de identidade, e demonstraram alta precisão para a extração de informações de citação. A primeira linguagem formalmente definida para os modelos de probabilidade de universo aberto foi a BLOG (Milch *et al*., 2005), que veio com um algoritmo de inferência completo (embora lento) CMMC para todos os modelos bem definidos. (O código do programa ligeiramente visível na capa deste livro é parte de um modelo BLOG para a detecção de

explosões nucleares a partir de sinais sísmicos, como parte do regime de verificação do UN Comprehensive Tet Ban Treaty.) Laskey (2008) descreve outra linguagem de modelagem de universo aberto chamada **redes bayesianas multientidades**.

Como explicamos no Capítulo 13, os primeiros sistemas probabilísticos perderam o interesse no início da década de 1970, deixando um vácuo parcial a ser preenchido por métodos alternativos. Os fatores de certeza foram criados para uso no sistema especialista médico MYCIN (Shortliffe, 1976), destinado a ser tanto uma solução de engenharia quanto um modelo de julgamento humano sob incerteza. A coleção *Rule-Based Expert Systems* (Buchanan e Shortliffe, 1984) fornece uma visão geral completa do MYCIN e de seus descendentes (veja também Stefik, 1995). David Heckerman (1986) mostrou que uma versão ligeiramente modificada de cálculos do fator de certeza fornece resultados probabilísticos corretos em alguns casos, mas resulta em séria superestimativa da evidência em outros casos. O sistema especialista PROSPECTOR (Duda *et al.*, 1979) usou uma abordagem baseada em regras, na qual as regras eram justificadas por uma suposição (raramente sustentável) de independência global.

A teoria de Dempster-Shafer se origina de um ensaio de Arthur Dempster (1968) que propôs uma generalização da probabilidade a valores de intervalos em uma regra de combinação para utilizá-los. Um trabalho posterior de Glenn Shafer (1976) levou à visão da teoria de Dempster-Shafer como uma abordagem concorrente para a probabilidade. Pearl (1988) e Ruspini *et al.* (1992) analisam o relacionamento entre a teoria de Dempster-Shafer e a teoria da probabilidade-padrão.

Os conjuntos difusos foram desenvolvidos por Lotfi Zadeh (1965) em resposta à dificuldade percebida de fornecer entradas exatas para sistemas inteligentes. O texto de Zimmermann (2001) fornece uma introdução completa à teoria de conjuntos difusos; os documentos sobre aplicações difusas estão reunidos em Zimmermann (1999). Como mencionamos no texto, a lógica difusa com frequência foi percebida incorretamente como uma concorrente direta da teoria da probabilidade, embora de fato ela focalize um conjunto de questões diferente. A **teoria da possibilidade** (Zadeh, 1978) foi introduzida para lidar com a incerteza em sistemas difusos e tem muito em comum com a probabilidade. Dubois e Prade (1994) fornecem um estudo completo das conexões entre a teoria da possibilidade e a teoria da probabilidade.

O ressurgimento da probabilidade dependia principalmente do desenvolvimento das redes bayesianas de Pearl como um método para representar e usar informações de independência condicional. Esse ressurgimento não veio sem luta; o combativo ensaio de Peter Cheeseman (1985), "In Defense of Probability" e seu artigo posterior "An Inquiry into Computer Understanding" (Cheeseman, 1988, com comentários) deram algum sabor ao debate. Eugene Charniak ajudou a apresentar as ideias para pesquisadores de IA com um artigo popular, "Bayesian networks without tears"[11] (1991), e um livro (1993). O livro de Dean e Wellman (1991) também ajudou a introduzir as redes bayesianas para os pesquisadores de IA. Uma das principais objeções dos logicistas foi o fato de que os cálculos numéricos que a teoria da probabilidade exigia não eram aparentes para a introspecção e presumiam um nível não realista de precisão em nosso conhecimento incerto. O desenvolvimento de **redes probabilísticas qualitativas** (Wellman, 1990a) forneceu uma abstração puramente qualitativa de redes bayesianas, usando a noção de influências positivas e negativas entre variáveis. Wellman mostra que, em muitos casos, tais informações são suficientes para a tomada de decisões ótimas sem a necessidade da especificação exata de valores de probabilidade. Goldszmidt

e Pearl (1996) têm uma abordagem semelhante. O trabalho de Adnan Darwiche e Matt Ginsberg (1992) extrai as propriedades básicas de condicionamento e combinação de evidências da teoria da probabilidade e mostra que elas também podem ser aplicadas em raciocínio lógico e em raciocínio default. Com frequência, os programas falam mais alto que as palavras, e a disponibilidade pronta do software de alta qualidade, como a ferramenta Net Bayes (Murphy, 2001), acelerou a adoção da tecnologia.

A publicação mais importante para o crescimento das redes bayesianas foi, sem dúvida, o texto *Probabilistic Reasoning in Intelligent Systems* (Pearl, 1988). Vários textos excelentes (Lauritzen, 1996; Jensen, 2001; Korb e Nicholson, 2003; Jensen, 2007; Darwiche, 2009; Koller e Friedman, 2009) fornecem tratamentos completos dos tópicos que cobrimos neste capítulo. Novas pesquisas sobre raciocínio probabilístico aparecem tanto em periódicos importantes sobre IA, como *Artificial Intelligence* e *Journal of AI Research*, quanto em periódicos mais especializados, como o *International Journal of Approximate Reasoning*.

Muitos artigos sobre modelos gráficos, que incluem as redes bayesianas, são publicados em periódicos especializados em estatística. Os anais das conferências sobre Uncertainty in Artificial Intelligence (UAI), Neural Information Processing Systems (NIPS) e Artificial Intelligence and Statistics (AISTATS) são excelentes fontes de pesquisa atual.

EXERCÍCIOS

14.1 Temos um saco com três moedas tendenciosas a, b e c e com probabilidade de 20%, 60% e 80%, respectivamente de virar cara. Uma moeda foi retirada aleatoriamente do saco (com probabilidade igual entre as três moedas) e lançada três vezes para gerar os resultados X_1, X_2 e X_3.

a. Desenhe a rede bayesiana correspondente a essa configuração e defina os TPCs necessários.

b. Calcule qual moeda é mais provável que tenha sido retirada do saco se os lançamentos observados deram cara duas vezes e coroa uma vez.

14.2 A equação 14.1 define a distribuição conjunta representada por uma rede bayesiana em termos dos parâmetros $\theta(X_i \mid Pais\ (X_i))$. Este exercício pede para obter a equivalência entre os parâmetros e as probabilidades condicionais $\mathbf{P}(X_i \mid Pais(X_i))$ dessa definição.

a. Considere uma rede simples $X \rightarrow Y \rightarrow Z$ com três variáveis booleanas. Utilize as Equações 13.3 e 13.6 para expressar a probabilidade condicional $P(z \mid y)$ como a razão entre duas somas, cada uma sobre as entradas na distribuição conjunta $\mathbf{P}(X, Y, Z)$.

b. Agora utilize a Equação 14.1 para escrever essa expressão em termos de parâmetros da rede $\theta(X)$, $\theta(Y \mid X)$ e $\theta(Z \mid Y)$.

c. Em seguida, expanda os somatórios da expressão da parte (b) escrevendo explicitamente os termos para os valores verdadeiros e falsos de cada variável somada. Assumindo que todos os parâmetros da rede satisfazem a restrição $\Sigma_{xi}\ \theta\ (x_i \mid Pais(X_i) = 1$, mostre que a expressão resultante se reduz para $\theta\ (x \mid y)$.

d. Generalize essa derivação para mostrar que $\theta\ (X_i|Pais(X_i) = \mathbf{P}(X_i|Pais\ (X_i))$ para qualquer rede

bayesiana.

14.3 A operação de **reversão de arco** em uma rede bayesiana permite-nos mudar a direção de um arco $X \rightarrow Y$, enquanto preserva a distribuição de probabilidade conjunta que a rede representa (Shachter, 1986). A reversão de arco pode exigir a introdução de novos arcos: todos os pais de X também se tornaram pais de Y, e todos os pais de Y se tornaram pais de X.

a. Suponha que X e Y comecem com os pais m e n, respectivamente, e que todas as variáveis têm valores k. Calculando a mudança no tamanho de TCPs de X e Y, mostre que o número total de parâmetros da rede não pode diminuir durante a reversão do arco. (*Dica*: Os pais de X e Y não precisam ser separados.)

b. Sob que circunstâncias o número total pode permanecer constante?

c. Faça $\mathbf{U} \cup \mathbf{V}$ os pais de X e $\mathbf{V} \cup \mathbf{W}$ os pais de Y, onde $\mathbf{U}$ e $\mathbf{W}$ são disjuntos. As fórmulas para os novos TPPs após a reversão de arco são as seguintes:

$$\mathbf{P}(Y \mid \mathbf{U}, \mathbf{V}, \mathbf{W}) = \sum_x \mathbf{P}(Y \mid \mathbf{V}, \mathbf{W}, x)\mathbf{P}(x \mid \mathbf{U}, \mathbf{V})$$
$$\mathbf{P}(X \mid \mathbf{U}, \mathbf{V}, \mathbf{W}, Y) = \mathbf{P}(Y \mid X, \mathbf{V}, \mathbf{W})\mathbf{P}(X \mid \mathbf{U}, \mathbf{V})/\mathbf{P}(Y \mid \mathbf{U}, \mathbf{V}, \mathbf{W}) .$$

Demonstre que a nova rede expressa a mesma distribuição disjunta sobre todas as variáveis como a da rede original.

14.4 Considere a rede bayesiana na Figura 14.2.

a. Se nenhuma evidência for observada, *Roubo* e *Terremoto* são independentes? Demonstre isso pela semântica numérica e pela semântica topológica.

b. Se observarmos que *Alarme* = *verdadeiro*, *Roubo* e *Terremoto* são independentes? Justifique sua resposta calculando se as probabilidades envolvidas satisfazem a definição de independência condicional.

14.5 Suponha que em uma rede bayesiana que contém uma variável não observada Y, todas as variáveis na cobertura de Markov CM(Y) foram observadas.

a. Demonstre que a remoção do nó Y da rede não irá afetar a distribuição posterior de qualquer outra variável não observada na rede.

b. Questione se podemos remover Y se estamos planejando usar (i) amostragem de rejeição (ii) ponderação de probabilidade.

14.6 Seja H_x uma variável aleatória que denota a lateralidade de um indivíduo x, com valores possíveis l ou r. Uma hipótese comum é que a lateralidade esquerda ou direita é herdada por um mecanismo simples; ou seja, talvez haja um gene G_x, também com valores l ou r, e talvez a lateralidade real acabe praticamente a mesma (com alguma probabilidade s) como sendo o gene que um indivíduo possui. Além disso, talvez, o gene tenha a mesma probabilidade de ser herdado de ambos os pais de um indivíduo, com pequena probabilidade m diferente de zero de uma mutação aleatória trocar a lateralidade.

a. Qual das três redes na Figura 14.20 afirma que $\mathbf{P}(G_{pai}, G_{mãe}, G_{filho}) = \mathbf{P}(G_{pai})\,\mathbf{P}(G_{mãe})\,\mathbf{P}(G_{filho})$?

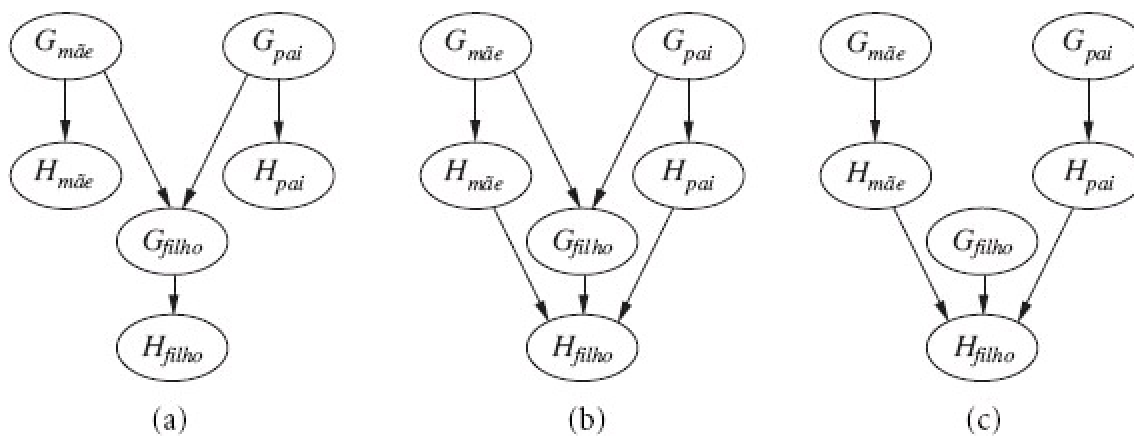


Figura 14.20 Três estruturas possíveis para uma rede bayesiana descrevendo a herança genética de lateralidade.

b. Qual das três redes faz afirmações de independência que são consistentes com a hipótese sobre a herança da lateralidade?

c. Qual das três redes é a melhor descrição da hipótese?

d. Escreva o TPC para o nó G_{filho} na rede (a), em termos de s e m.

e. Suponha que $P(G_{\text{pai}} = l) = P(G_{\text{mãe}} = l) = \theta$. Na rede (a), derive uma expressão para $P(G_{\text{filho}} = l)$ em termos de m e θ apenas, por condicionamento aos nós do seu pai.

f. Em condições de equilíbrio genético, esperamos que a distribuição de genes seja a mesma através das gerações. Utilize essa opção para calcular o valor de θ e, dado o que você sabe sobre a lateralidade em seres humanos, explique por que a hipótese descrita no início desta questão deve estar errada.

14.7 A **cobertura de Markov** de uma variável foi definida neste capítulo. Prove que uma variável é independente de todas as outras variáveis na rede, dada sua cobertura de Markov, e derive a Equação 14.12.

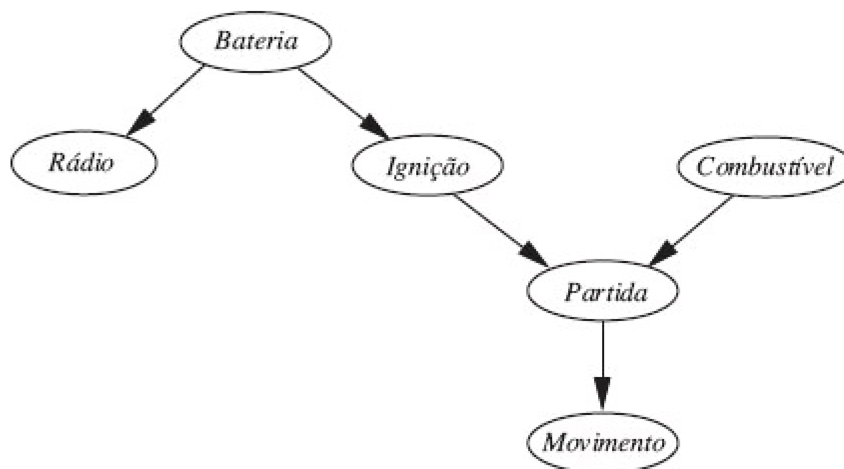


Figura 14.21 Uma rede bayesiana que descreve algumas características do sistema elétrico e domotor de um carro. Cada variável é booleana, e o valor *verdadeiro* indica que o aspecto correspondente do veículo está em perfeita ordem.

14.8 Considere a rede para diagnóstico de automóveis mostrada na Figura 14.21.

a. Estenda a rede com as variáveis booleanas *TempoGelado* e *MotorDePartida*.

b. Apresente tabelas de probabilidade condicional razoáveis para todos os nós.

c. Quantos valores independentes estão contidos na distribuição de probabilidade conjunta para oito nós booleanos, supondo-se que não seja conhecida nenhuma relação de independência condicional válida entre eles?

d. Quantos valores de probabilidade independentes contêm suas tabelas de rede?

e. A distribuição condicional para *Partida* poderia ser descrita como uma distribuição **E ruidosa**. Defina essa família em geral e relacione-a com a distribuição OU ruidosa.

14.9 Considere a família de redes gaussianas lineares, ilustrada na página 520.

a. Em uma rede de duas variáveis, seja X_1 o pai de X_2, sendo que X_1 tem um *a priori* gaussiano, e seja $\mathbf{P}(X_2 \mid X_1)$ uma distribuição gaussiana linear. Mostre que a distribuição conjunta $P(X_1,X_2)$ é um gaussiano multivariado e calcule sua matriz de covariância.

b. Prove por indução que a distribuição conjunta para uma rede gaussiana linear geral sobre $X_1,\ldots, X_n$ também é um gaussiano multivariado.

14.10 A distribuição probit definida neste capítulo descreve a distribuição de probabilidade para um filho booleano, dado um único pai contínuo.

a. Como a definição poderia ser estendida para cobrir vários pais contínuos?

b. Como ela poderia ser estendida para lidar com uma variável filha *multivalorada*? Considere tanto casos em que os valores do filho estão ordenados (como na seleção de uma marcha enquanto se está dirigindo, dependendo de velocidade, do declive, da aceleração desejada etc.) quanto casos em que eles não estão ordenados (como na seleção de ônibus, trem ou carro para chegar ao trabalho). [*Sugestão*: Considere modos de dividir os valores possíveis em dois conjuntos, a fim de imitar uma variável booleana.]

14.11 Em sua estação de energia nuclear local, existe um alarme que detecta quando um indicador de temperatura excede dado limiar. O indicador mede a temperatura do núcleo. Considere as variáveis booleanas A (o alarme soa), F_A (alarme defeituoso) e F_G (medidor defeituoso) e os nós de vários valores G (leitura do medidor) e T (temperatura real do núcleo).

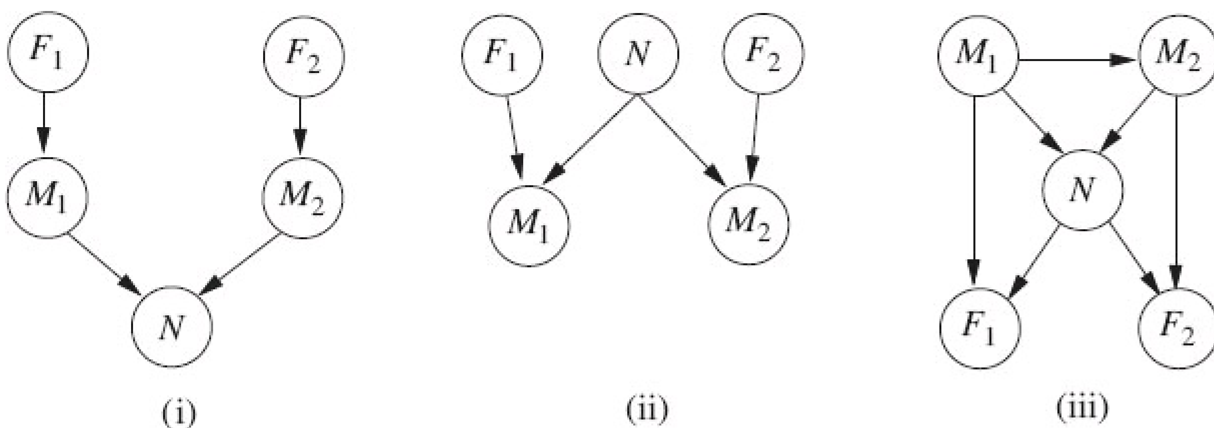


Figura 14.22 Três redes possíveis para o problema do telescópio.

a. Trace uma rede bayesiana para esse domínio considerando que o medidor tem maior probabilidade de falhar quando a temperatura do núcleo fica muito alta.

b. Sua rede é uma poliárvore? Por quê?

c. Suponha que existam apenas duas temperaturas reais e medidas possíveis, normal e alta; a probabilidade de que o medidor forneça a temperatura correta é x quando ele está funcionando, mas é y quando ele apresenta defeito. Forneça a tabela de probabilidade condicional associada a G.

d. Suponha que o alarme funcione corretamente, a menos que esteja defeituoso e, nesse caso, ele nunca tocará. Forneça a tabela de probabilidade condicional associada com A.

e. Suponha que o alarme e o medidor estejam funcionando e que o alarme toque. Calcule uma expressão para a probabilidade de que a temperatura do núcleo esteja muito alta, em termos das várias probabilidades condicionais na rede.

14.12 Dois astrônomos em diferentes partes do mundo fazem medições M_1 e M_2 do número de estrelas N em alguma região pequena do céu, usando seus telescópios. Normalmente, existe uma pequena possibilidade e de erro de até uma estrela em cada direção. Cada telescópio também pode (com uma probabilidade f, muito menor) estar completamente fora de foco (eventos F_1 e F_2), e nesse caso o cientista vai contar três ou mais estrelas a menos (ou, se N for menor que 3, deixar de detectar quaisquer estrelas). Considere as três redes mostradas na Figura 14.22.

a. Quais dessas redes bayesianas são representações corretas (mas não necessariamente eficientes) das informações precedentes?

b. Qual é a melhor rede? Explique.

c. Descreva uma distribuição condicional para $\mathbf{P}(M_1 \mid N)$, para o caso em que $N \in \{1, 2, 3\}$ e $M_1 \in \{0, 1, 2, 3, 4\}$. Cada entrada na distribuição condicional deve ser expressa como uma função dos parâmetros e e/ou f.

d. Suponha $M_1 = 1$ e $M_2 = 3$. Quais são os números *possíveis* de estrelas se não supusermos nenhuma restrição *a priori* sobre os valores de N?

e. Qual é o número *mais provável* de estrelas, dadas essas observações? Explique como calcular esse número ou, se não for possível calculá-lo, explique que informações adicionais são necessárias e como elas afetariam o resultado.

14.13 Considere a rede mostrada na Figura 14.22 (ii) e assuma que dois telescópios trabalham de forma idêntica.

$$N \in \{1, 2, 3\} \text{ e } M_1, M_2 \in \{0, 1, 2, 3, 4\},$$

com os TPCs simbólicos como descrito no exercício 14.12. Usando o algoritmo de enumeração (Figura 14.9), calcule a distribuição de probabilidade $P(N|M_1 = 2, M_2 = 2)$.

14.14 Considere a rede de Bayes mostrada na Figura 14.23.

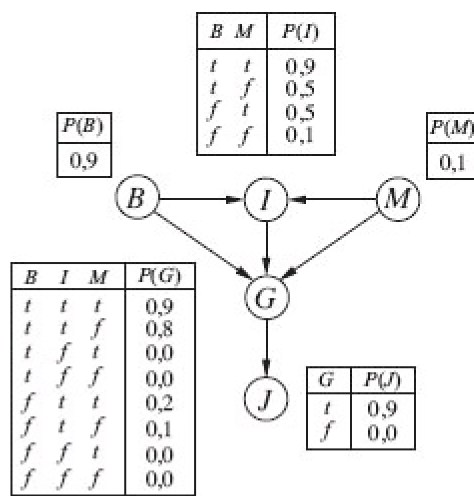


Figura 14.23 Uma rede de Bayes simples com variáveis booleanas B = *InfringiuLeiEleitoral*, I = *Indiciado*, M = *PromotorPoliticamenteMotivado*, G = *ConsideradoCulpado*, J = *Preso*.

a. Quais dos seguintes itens são declarados pela *estrutura* de rede?

(i) $\mathbf{P}(B, I, M) = \mathbf{P}(B)\mathbf{P}(I)\mathbf{P}(M)$.

(ii) $\mathbf{P}(J \mid G) = \mathbf{P}(J \mid G, I)$.

(iii) $\mathbf{P}(M \mid G, B, I) = \mathbf{P}(M \mid G, B, I, J)$.

b. Calcule o valor de $P(b, i, \neg m, g, j)$.

c. Calcule a probabilidade de alguém ir para a cadeia uma vez que infringiu a lei, foi indiciado e enfrentou um promotor politicamente motivado.

d. Uma **independência específica de contexto** permite que uma variável seja independente de alguns de seus pais, dados certos valores de outros. Além das independências condicionais usuais, dadas pela estrutura de grafo, que independência específica de contexto existe na rede de Bayes na Figura 14.23?

e. Suponha que queiramos adicionar a variável P = *PerdãoPresidencial* à rede; desenhe a nova rede e explique brevemente os links que adicionou.

14.15 Considere o algoritmo de eliminação de variáveis na Figura 14.11.

a. A Seção 14.4 aplica a eliminação de variáveis à consulta

$\mathbf{P}(Roubo \mid JoãoLiga = verdadeiro, MariaLiga = verdadeiro)$.

Execute os cálculos indicados e verifique se a resposta está correta.

b. Conte o número de operações aritméticas executadas e compare-o com o número de operações executadas pelo algoritmo de enumeração.

c. Suponha que uma rede tenha a forma de uma *cadeia*: uma sequência de variáveis booleanas X_1, …, X_n onde $Pais(X_i) = \{X_{i-1}\}$ para $i = 2, \ldots, n$. Qual é a complexidade da computação de $\mathbf{P}(X_1 \mid Xn = verdadeiro)$ usando enumeração? E usando eliminação de variáveis?

d. Prove que a complexidade de execução da eliminação de variáveis sobre uma rede de poliárvore é linear no tamanho da árvore para qualquer ordenação de variáveis consistente com a estrutura da rede.

14.16 Investigue a complexidade da inferência exata em redes bayesianas gerais:

a. Prove que qualquer problema de 3-SAT pode ser reduzido à inferência exata em uma rede bayesiana construída para representar o problema específico e, consequentemente, que a inferência exata é NP-difícil. [*Sugestão*: Considere uma rede com uma variável para cada símbolo de proposição, uma para cada cláusula e uma para a conjunção de cláusulas.]

b. O problema de contar o número de atribuições de satisfação para um problema de 3-SAT é #P-completo. Mostre que a inferência exata tem pelo menos a mesma dificuldade.

14.17 Considere o problema de gerar uma amostra aleatória a partir de uma distribuição especificada sobre uma única variável. Suponha que um gerador de números aleatórios esteja disponível e retorne um número aleatório uniformemente distribuído entre 0 e 1.

a. Seja X uma variável discreta com $P(X = x_i) = p_i$ para $i \in \{1,\ldots, k\}$. A **distribuição cumulativa** de X fornece a probabilidade de $X \in \{x_1,\ldots, x_j\}$ para cada j possível (veja também o Apêndice A). Explique como calcular a distribuição cumulativa no tempo $O(k)$ e como gerar uma única amostra de X a partir dela. Esta última operação pode ser realizada em tempo menor que $O(k)$?

b. Suponha agora que queiramos gerar N amostras de X, onde $N \gg k$. Explique como fazer isso com um tempo de execução esperado por amostra que seja *constante* (isto é, independente de k).

c. Agora, considere uma variável com valores contínuos que tenha uma distribuição parametrizada (por exemplo, gaussiana). Como é possível gerar amostras a partir de tal distribuição?

d. Suponha que você queira consultar uma variável de valores contínuos e esteja usando um algoritmo de amostragem como PONDERAÇÃO-DE-PROBABILIDADE para realizar a inferência. De que maneira você teria de modificar o processo de responder à consulta?

14.18 Considere a consulta **P**(*Chuva* | *Irrigador* = *verdadeiro*, *GramaMolhada* = *verdadeiro*) na Figura 14.12(a) (página 529) e como o CMMC pode responder à consulta.

a. Quantos estados tem a cadeia de Markov?

b. Calcule a **matriz de transição Q** que contém $\theta(\mathbf{y} \rightarrow \mathbf{y}')$ para todo **y**, **y'**.

c. O que representa $\mathbf{Q}^2$, o quadrado da matriz de transição?

d. O que representa $\mathbf{Q}^n$ à medida que $n \rightarrow \infty$?

e. Explique como realizar a inferência probabilística em redes bayesianas supondo que $\mathbf{Q}^n$ esteja disponível. Esse é um caminho prático de realizar a inferência?

14.19 Este exercício explora a distribuição estacionária de métodos de amostragem de Gibbs.

a. A composição convexa $[\alpha, \theta_1, 1-\alpha, \theta_2]$ de θ_1 e θ_2 é uma distribuição de probabilidade de transição que primeiro escolhe um de θ_1 e θ_2 com probabilidades α e $1-\alpha$, respectivamente, e, em seguida, aplica-se o que for escolhido. Demonstre que, se θ_1 e θ_2 estão em equilíbrio detalhado com π, sua composição também é convexa em equilíbrio detalhado com π. (Observação: esse resultado justifica uma variante do ASK-GIBBS em que as variáveis são escolhidas de forma aleatória, em vez de amostradas em uma sequência fixa.)

b. Prove que, se cada um de θ_1 e θ_2 tem π como distribuição estacionária, a composição

sequencial $\theta = \theta_1 \circ \theta_2$ também tem π como distribuição estacionária.

14.20 O algoritmo de **Metropolis-Hastings** é um membro da família CMMC e, como tal, é projetado para gerar amostras **x** (eventualmente) de acordo com probabilidades-alvo π(**x**) (normalmente estamos interessados em amostragem de π(**x**) = P(**x** | **e**)). Assim como a têmpora simulada, o Metropolis-Hastings opera em duas etapas. Primeiro, faz amostra de um novo estado **x′** a partir de uma proposta de **distribuição de proposta** $\theta(\mathbf{x}'|\mathbf{x})$, dado o atual estado **x**. Então, probabilisticamente, aceita ou rejeita **x′** de acordo com a **probabilidade de aceitação**

$$\alpha(\mathbf{x}' \mid \mathbf{x}) = \min\left(1, \frac{\pi(\mathbf{x}')q(\mathbf{x} \mid \mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{x}' \mid \mathbf{x})}\right) .$$

Se a proposta for rejeitada, o estado continua em **x**.

a. Considere uma etapa de amostragem de Gibbs comum para uma variável X_i específica. Mostre que essa etapa, considerada como uma proposta, é a garantia de ser aceita pelo Metropolis-Hastings (assim, a amostragem de Gibbs é um caso especial de Metropolis-Hastings).

b. Mostre que o processo de duas etapas anterior, visto como distribuição de probabilidade de transição, está em equilíbrio detalhado com π.

14.21 Três times de futebol, A, B e C, jogam entre si uma vez. Cada partida ocorre entre duas equipes, e cada uma pode vencer, empatar ou perder. Cada time tem um grau fixo e desconhecido de qualidade — um inteiro que varia de 0 a 3 — e o resultado de uma partida depende probabilisticamente da diferença de qualidade entre os dois times.

a. Construa um modelo de probabilidade relacional para descrever esse domínio e sugira valores numéricos para todas as distribuições de probabilidade necessárias.

b. Construa a rede bayesiana equivalente para as três partidas.

c. Suponha que, nas duas primeiras partidas, A vença B e empate com C. Empregando um algoritmo de inferência exata de sua escolha, calcule a distribuição posterior para o resultado da terceira partida.

d. Suponha que existam n times na liga e que temos os resultados para todas as partidas, com exceção da última. Como a complexidade de prognosticar o resultado do último jogo varia com n?

e. Investigue a aplicação de CMMC para esse problema. Com que rapidez ele converge na prática e com que facilidade ele aumenta sua escala?

---

[1] Esse é o nome mais comum, mas existem muitos sinônimos, como **rede de crença**, **rede probabilística**, **rede causal** e **mapa de conhecimento**. Em estatística, a expressão **modelo gráfico** se refere a uma classe um pouco mais ampla, que inclui as redes bayesianas. Uma extensão de redes bayesianas chamada **rede de decisão** ou **diagrama de influência** será focalizada no Capítulo 16.

[2] Também existe um critério topológico geral chamado **separação d** para se decidir se um conjunto de nós **X** é independente condicionalmente de outro conjunto **Y**, dado um terceiro conjunto **Z**. O critério é bastante complicado e não é necessário para derivar os algoritmos deste capítulo; assim, vamos omiti-lo. Os detalhes podem ser encontrados em Pearl (1988) ou Darwiche (2009). Shachter (1998) fornece um método mais intuitivo de averiguar a separação d.

[3] Segue-se que a inferência em redes gaussianas lineares demora apenas o tempo $O(n^3)$ no pior caso, independentemente da topologia da rede. Na Seção 14.4, veremos que a inferência para redes de variáveis discretas é NP-difícil.

[4] Uma expressão como $\Sigma_e\, P(a,\ e)$ significa efetuar o somatório de $P(A = a, E = e)$ para todos os valores possíveis de e. Existe uma ambiguidade no fato de $P(e)$ ser usada para indicar tanto $P(E= verdadeira)$ quanto $P(E = e)$, mas deve ficar claro a partir do contexto o que se pretende dizer; em particular, no contexto de um somatório, a última forma é a pretendida.

[5] No caso ideal, seria interessante usar uma distribuição de amostragem igual à distribuição posterior verdadeira $P(\mathbf{z} \mid \mathbf{e})$, para levar em conta toda a evidência. Porém, isso não pode ser feito de modo eficiente. Se pudesse, seria possível realizar uma aproximação da probabilidade desejada até uma precisão arbitrária com um número de amostras polinomial. Pode-se demonstrar que não é possível existir tal esquema de aproximação em tempo polinomial.

[6] Um teórico dos jogos aconselharia um cliente desonesto a evitar a detecção, recomendando ocasionalmente um bom livro de um concorrente. Consulte o Capítulo 17.

[7] O nome do *modelo de probabilidade relacional* foi dado por Pfeiffer (2000) para uma representação ligeiramente diferente, mas as ideias subjacentes são as mesmas.

[8] Algumas condições técnicas devem ser observadas para garantir que o MPR defina uma distribuição adequada. Em primeiro lugar, as dependências devem ser *acíclicas*; caso contrário, a rede bayesiana resultante terá ciclos e não definirá uma distribuição adequada. Segundo, as dependências devem ser *bem fundamentadas*, isto é, não pode haver correntes de ancestrais infinitos, tais como poderia surgir a de dependências recursivas. Sob algumas circunstâncias (ver Exercício 14.6), um cálculo de ponto fixo produz um modelo de probabilidade bem definida para um MPR recursivo.

[9] A distribuição $LogNormal\,[\mu, \Sigma^2]\,(x)$ é equivalente a uma distribuição $N[\mu, \sigma^2]\,(x)$ sobre $\log_e(x)$.

[10] I. J. Good foi o principal estatístico da equipe de quebra de códigos de Turing na Segunda Guerra Mundial. Em *2001: Uma Odisseia no Espaço* (Clarke, 1968a), Good e Minsky recebem o crédito pela inovadora criação que levou ao desenvolvimento do computador HAL 9000.

[11] O título da versão original do artigo era “Pearl for swine”.

CAPÍTULO 15

Raciocínio probabilístico temporal

Em que tentamos interpretar o presente, entender o passado e talvez prever o futuro, ainda que muito pouco seja transparente.

Os agentes em ambientes parcialmente observáveis devem ser capazes de manter o controle do estado atual até a medida que seus sensores permitam. Na Seção 4.4 mostramos uma metodologia para fazer isso: um agente mantinha um **estado de crença** que representava quais os estados do mundo eram possíveis. Com base no estado de crença e em um **modelo de transição**, o agente pode prever como o mundo pode evoluir na próxima etapa de tempo. Com base nas percepções observadas e em um **modelo de sensores**, o agente pode atualizar o estado de crença. Essa é uma ideia sempre presente: no Capítulo 4, os estados de crença eram representados por conjuntos de estados explicitamente enumerados, enquanto nos Capítulos 7 e 11 eram representados por fórmulas lógicas. Essas abordagens definiram os estados de crença em termos de quais estados do mundo eram *possíveis,* mas não podiam afirmar nada sobre quais estados eram *prováveis* ou *improváveis*. Neste capítulo, usamos a teoria da probabilidade para quantificar o grau de crença em elementos do estado de crença.

Como mostramos na Seção 15.1, o próprio tempo é tratado da mesma forma como no Capítulo 7: um mundo em mudança é modelado com a utilização de uma variável aleatória para cada aspecto do estado do mundo *em cada instante no tempo*. A transição e os modelos sensoriais podem ser incertos: o modelo de transição descreve a distribuição de probabilidade das variáveis no tempo t, dado o estado do mundo em tempos passados, enquanto o modelo de sensores descreve a probabilidade de cada percepção no tempo t, dado o estado do mundo atual. A Seção 15.2 define as tarefas básicas de inferência e descreve a estrutura geral de algoritmos de inferência para modelos temporários. Em seguida, descrevemos três tipos específicos de modelos: **modelos ocultos de Markov**, **filtros de Kalman** e **redes bayesianas dinâmicas** (que incluem modelos ocultos de Markov e filtros de Kalman como casos especiais). Finalmente, a Seção 15.6 examina os problemas encontrados quando se mantém o controle de mais de uma coisa.

15.1 TEMPO E INCERTEZA

Desenvolvemos nossas técnicas para raciocínio probabilístico no contexto de mundos *estáticos*,

nos quais cada variável aleatória tem um único valor fixo. Por exemplo, ao consertar um carro, supomos que qualquer peça que esteja danificada continuará danificada durante o processo de diagnóstico; nosso trabalho é deduzir o estado do automóvel a partir da evidência observada, que também permanece fixa.

Agora, considere um problema ligeiramente diferente: tratar um paciente diabético. Como no caso do conserto de automóveis, temos evidências, como doses de insulina recentes, alimentação, medições de açúcar no sangue e outros sinais físicos. A tarefa é avaliar o estado atual do paciente, inclusive o nível real de açúcar no sangue e o nível de insulina. Dadas essa informações, podemos tomar uma decisão sobre a alimentação e a dose de insulina do paciente. Diferentemente do caso de conserto de automóveis, aqui os aspectos *dinâmicos* do problema são essenciais. Os níveis de açúcar no sangue e as medições subsequentes podem mudar rapidamente com o tempo, dependendo da alimentação e das doses de insulina recentes do paciente, de sua atividade metabólica, da hora do dia, e assim por diante. Para avaliar o estado atual a partir do histórico de evidência e prever os resultados de ações de tratamento, devemos modelar essas mudanças.

As mesmas considerações surgem em muitos outros contextos, tal como acompanhar a localização do robô, o controle da atividade econômica de uma nação, até dar sentido a uma sequência de palavras faladas. Como é possível modelar situações dinâmicas como essas?

15.1.1 Estados e observações

Vemos o mundo como uma série de instantâneos, ou **fatias de tempo**, cada uma das quais contém um conjunto de variáveis aleatórias, algumas observáveis, e outras, não.[1] Por simplicidade, vamos supor que o mesmo subconjunto de variáveis seja observável em cada fatia de tempo (embora isso não seja estritamente necessário em nada do que será discutido a seguir). Usaremos $\mathbf{X}_t$ para denotar o conjunto de variáveis de estados no tempo *t,* que assumimos ser não observáveis, e $\mathbf{E}_t$ para denotar o conjunto de variáveis de evidência observáveis. A observação no tempo t é $\mathbf{E}_t = \mathbf{e}_t$ para algum conjunto de valores $\mathbf{e}_t$.

Considere o exemplo a seguir: suponha que você seja o guarda de segurança em alguma instalação subterrânea secreta. Você quer saber se hoje está chovendo, mas seu único acesso ao mundo exterior ocorre a cada manhã, quando vê o diretor entrando com ou sem guarda-chuva. Para cada dia *t*, o conjunto $\mathbf{E}_t$ contém portanto uma única variável de evidência *Guarda-chuvat* ou G_t para abreviar (indicando se o guarda-chuva aparece ou não) e o conjunto $\mathbf{X}_t$ contém uma única variável de estado $Chuva_t$ ou C_t (indicando se está chovendo). Outros problemas podem envolver conjuntos maiores de variáveis. No exemplo do diabetes, poderíamos ter variáveis de evidência como $AçúcarNoSangueMedido_t$ e $Pulsação_t$, e variáveis de estados como $AçúcarNoSangue_t$ e $ConteúdoDoEstômago_t$. (Observe que $AçúcarNoSangue_t$ e $MedidaAçúcarNoSangue_t$ não são as mesmas variáveis; essa é a forma como tratamos com medidas ruidosas de quantidades reais.)

O intervalo entre fatias de tempo também depende do problema. Para monitoramento de diabetes, um intervalo apropriado poderia ser uma hora, em vez de um dia. Neste capítulo, assumiremos o

intervalo entre fatias como fixo; assim poderemos rotular tempos como inteiros. Presumiremos que a sequência de estados se inicie em $t = 0$; por diversas razões que não nos interessam no momento, vamos supor que a evidência comece a chegar em $t = 1$ e não em $t = 0$. Consequentemente, nosso mundo de guarda-chuva é representado por variáveis de estados R_0, R_1, R_2,… e por variáveis de evidência U_1,U_2,… Usaremos a notação a:b para denotar a sequência de inteiros de a até b (inclusive) e a notação $\mathbf{X}_{a:b}$ para denotar o conjunto correspondente de variáveis desde $\mathbf{X}_A$ até $\mathbf{X}_B$. Por exemplo, $U_{1:3}$ corresponde às variáveis U_1, U_2, U_3.

15.1.2 Modelos de transição e de sensores

Tendo decidido o conjunto de variáveis de estado e evidência para dado problema, a próxima etapa é especificar como o mundo evolui (o modelo de transição) e como as variáveis de evidência obtêm seus valores (o modelo de sensores).

O modelo de transição especifica a distribuição de probabilidade sobre as variáveis de estado mais recente, dados os valores anteriores, ou seja, $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{0:t-1})$. Agora enfrentamos um problema: o conjunto $\mathbf{X}_{0:\,t-1}$ é ilimitado em tamanho à medida que t aumenta. Resolvemos o problema fazendo uma **suposição de Markov** — que o estado atual depende apenas de *um número fixo finito* de estados anteriores. Os processos que satisfazem essa suposição foram estudados em profundidade inicialmente pelo estatístico russo Andrei Markov (1856−1922) e são chamados de **processos de Markov ou cadeias de Markov**. Há vários tipos de processos; o mais simples é o **processo de Markov de primeira ordem**, em que o estado atual depende apenas do estado anterior e não de quaisquer estados anteriores. Em outras palavras, um estado fornece informação suficiente para tornar o futuro condicionalmente independente do passado, e temos

$$\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1}). \qquad (15.1)$$

Assim, em um processo de Markov de primeira ordem, o modelo de transição é a distribuição condicional $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1})$. O modelo de transição para um processo de Markov de segunda ordem é a distribuição condicional $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-2}, \mathbf{X}_{t-1})$. A Figura 15.1 mostra as estruturas de redes bayesianas correspondentes a processos de Markov de primeira e segunda ordem.

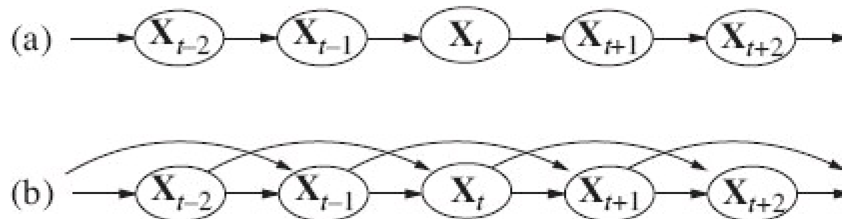


Figura 15.1 (a) Estrutura de rede bayesiana correspondente a um processo de Markov de primeira ordem como estado definido pelas variáveis $\mathbf{X}_t$. (b) Um processo de Markov de segunda ordem.

Mesmo com a suposição de Markov ainda há um problema: existem infinitos valores possíveis de t. Precisamos especificar uma distribuição diferente para cada etapa de tempo? Evitaremos esse

problema assumindo que as mudanças no estado do mundo são causadas por um **processo estacionário**, isto é, um processo de mudança que é governado por leis que não se alteram ao longo do tempo (não confunda *estacionário* com *estático*: em um processo *estático*, o próprio estado não muda). No mundo do guarda-chuva, então, a probabilidade condicional de chuva, $\mathbf{P}(R_t \mid R_{t-1})$, é a mesma para todo t, e temos apenas que especificar uma tabela de probabilidade condicional.

Agora para o modelo de sensores. As variáveis de evidência $\mathbf{E}_t$ poderiam depender de variáveis anteriores, bem como as variáveis de estado atual, mas qualquer estado que seja eficiente deve ser suficiente para gerar os valores de sensor correntes. Assim, fazemos uma **suposição de sensores de Markov** como segue:

$$\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t). \tag{15.2}$$

Assim, $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$ é o nosso modelo de sensor (às vezes chamado de **modelo de observação**). A Figura 15.2 mostra tanto o modelo de transição como o modelo de sensor para o exemplo do guarda-chuva. Observe a direção da dependência entre estado e sensores: as setas vão desde o estado real do mundo até os valores do sensor, pois o estado do mundo faz com que os sensores assumam valores particulares: a chuva *faz com que* o guarda-chuva apareça. (É claro que o processo de inferência se dá no sentido contrário; a distinção entre o sentido de dependências modeladas e o sentido da inferência é uma das principais vantagens das redes bayesianas.)

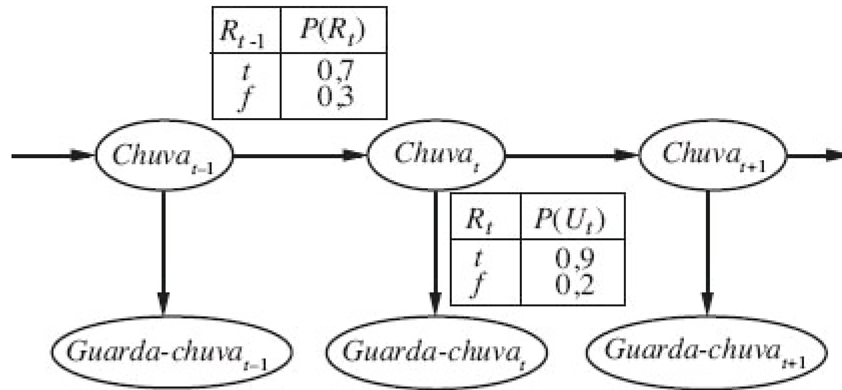


Figura 15.2 A estrutura de rede bayesiana e as distribuições condicionais que descrevem o mundo de guarda-chuva. O modelo de transição é $P(Chuva_t \mid Chuva_{t-1})$ e o modelo de sensores é $P(Guarda\text{-}chuva_t \mid Chuva_t)$.

Além de especificar os modelos de transição e de sensores, precisamos dizer como tudo iniciou — a distribuição de probabilidade anterior no tempo 0, $\mathbf{P}(\mathbf{X}_0)$. Com isso, temos uma especificação da distribuição conjunta completa sobre todas as variáveis, usando a Equação 14.2. Para qualquer t,

$$\mathbf{P}(\mathbf{X}_{0:t}, \mathbf{E}_{1:t}) = \mathbf{P}(\mathbf{X}_0) \prod_{i=1}^{t} \mathbf{P}(\mathbf{X}_i \mid \mathbf{X}_{i-1})\, \mathbf{P}(\mathbf{E}_i \mid \mathbf{X}_i)\,. \tag{15.3}$$

Os três termos do lado direito são o modelo do estado inicial $\mathbf{P}(\mathbf{X}_0)$, o modelo de transição $\mathbf{P}(\mathbf{X}_i \mid \mathbf{X}_{i-1})$ e o modelo de sensores $\mathbf{P}(\mathbf{E}_i \mid \mathbf{X}_i)$.

A estrutura na Figura 15.2 supõe um processo de Markov de primeira ordem porque se supõe que

a probabilidade de chuva dependa apenas de ter chovido ou não no dia anterior. O fato de tal hipótese ser razoável depende do próprio domínio. A hipótese de Markov de primeira ordem afirma que as variáveis de estados contêm *todas* as informações necessárias para caracterizar a distribuição de probabilidade para a próxima fatia de tempo. Às vezes, a hipótese é exatamente verdadeira — por exemplo, se uma partícula estiver executando um percurso aleatório ao longo do eixo x, mudando sua posição em ±1 a cada período de tempo e depois usando a coordenada x à medida que o estado fornece um processo de Markov de primeira ordem. Outras vezes, a hipótese é apenas aproximada, como no caso da previsão de chuva apenas com base no fato de ter chovido ou não no dia anterior. Há duas formas de melhorar a exatidão dessa aproximação:

1. Aumentar a ordem do modelo de processo de Markov. Por exemplo, poderíamos criar um modelo de segunda ordem adicionando $Chuva_{t-2}$ como um pai de $Chuva_t$, o que poderia resultar em previsões um pouco mais precisas. Por exemplo, em Palo Alto, Califórnia, raramente chove mais do que dois dias seguidos.
2. Aumentar o conjunto de variáveis de estados. Por exemplo, poderíamos adicionar $Estação_t$ para nos permitir incorporar registros históricos de estações chuvosas ou poderíamos adicionar $Temperatura_t$, $Umidade_t$ e $Pressão_t$ (talvez em uma série de locais) para nos dar a possibilidade de usar um modelo físico de condições chuvosas.

O Exercício 15.1 pede para mostrar que a primeira solução — aumentar a ordem — sempre pode ser reformulada como um aumento no conjunto de variáveis de estados, mantendo-se a ordem fixa. Observe que a adição de variáveis de estados poderia melhorar a capacidade de previsão do sistema, mas também aumentaria os *requisitos* de previsão: agora, temos de prever também as novas variáveis. Desse modo, estamos procurando por um conjunto de variáveis "autossuficiente", o que na realidade significa que temos de entender a "física" do processo que está sendo modelado. O requisito de modelagem precisa do processo obviamente será reduzido se pudermos adicionar novos sensores (por exemplo, medições de temperatura e pressão) que forneçam diretamente informações sobre as novas variáveis de estados.

Por exemplo, considere o problema de acompanhar um robô que vaga ao acaso no plano X−Y. Seria possível propor que a posição e a velocidade fossem um conjunto suficiente de variáveis de estados: alguém poderia simplesmente usar as leis de Newton para calcular a nova posição, e a velocidade poderia mudar de forma imprevisível. Porém, se o robô for alimentado por bateria, o esgotamento da carga da bateria tenderá a ter um efeito sistemático sobre a mudança na velocidade. Tendo em vista que isso, por sua vez, depende da quantidade de energia utilizada por todas as manobras anteriores, a propriedade de Markov é violada. Podemos restaurar a propriedade de Markov incluindo o nível de carga $Bateria_t$ como uma das variáveis de estados que compõem $\mathbf{X}_t$. Isso ajuda a prever o movimento do robô, mas, por outro lado, exige um modelo para prever $Bateria_t$ a partir de $Bateria_{t-1}$ e da velocidade. Em alguns casos, isso pode ser feito de forma confiável; percebemos que o erro se acumula através do tempo. Nesse caso, a exatidão será melhorada *adicionando-se um novo sensor* para indicar o nível de carga da bateria.

15.2 INFERÊNCIA EM MODELOS TEMPORAIS

Tendo configurado a estrutura de um modelo temporal genérico, podemos formular as tarefas básicas de inferência que devem ser resolvidas:

- **Filtragem:** Essa é a tarefa que consiste em calcular o **estado de crença** — a distribuição posterior sobre o estado mais recente — dada toda a evidência até o momento. A filtragem[2] é também chamada de **estimativa de estado**. Em nosso exemplo, desejamos calcular $\mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$. No exemplo do guarda-chuva, isso significaria calcular a probabilidade de chuva hoje, dadas todas as observações do portador de guarda-chuva feitas até agora. A filtragem é o que um agente racional precisa fazer, a fim de manter o controle do estado atual, de forma que possam ser tomadas decisões racionais. Ocorre que um cálculo quase idêntico fornece a **probabilidade** da sequência de evidências $P(\mathbf{e}_{1:t})$.
- **Previsão:** Essa é a tarefa de calcular a distribuição posterior sobre o estado *futuro*, dada toda a evidência até o momento. Ou seja, desejamos calcular $\mathbf{P}(\mathbf{X}_{t+k} \mid \mathbf{e}_{1:t})$ para algum $k > 0$. No exemplo de guarda-chuva, isso poderia significar calcular a probabilidade de chuva daqui a três dias, dadas todas as observações do portador de guarda-chuva feitas até agora. A previsão é útil para avaliar cursos de ação possíveis baseados nos resultados esperados.
- **Suavização**: Essa é a tarefa de calcular a distribuição posterior sobre um estado *passado* dada toda a evidência até o presente. Isto é, desejamos calcular $\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t})$ para algum k tal que $0 \leq k < t$. No exemplo do guarda-chuva, isso pode significar o cálculo da probabilidade de ter chovido na quarta-feira passada, dadas todas as observações do portador de guarda-chuva feitas até hoje. A suavização fornece uma estimativa melhor do estado do que a estimativa que estava disponível na época porque incorpora uma evidência maior.[3]
- **Explicação mais provável**: Dada uma sequência de observações, poderíamos desejar encontrar a sequência de estados que mais provavelmente gerou tais observações. Isto é, desejamos calcular $\operatorname{argmax}_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t} \mid \mathbf{e}_{1:t})$. Por exemplo, se o guarda-chuva aparecer em cada um dos três primeiros dias e estiver ausente no quarto dia, a explicação mais provável será a de que choveu nos três primeiros dias e não choveu no quarto dia. Os algoritmos para essa tarefa são úteis em muitas aplicações, inclusive no reconhecimento da fala — em que o objetivo é descobrir a sequência de palavras mais provável, dada uma série de sons — e na reconstrução de cadeias de bits transmitidos sobre um canal ruidoso.

Além dessas tarefas de inferência, temos também

- **Aprendizagem**: Os modelos de transição e de sensores, se ainda não são conhecidos, podem ser aprendidos de observações. Da mesma maneira que nas redes bayesianas estáticas, o aprendizado de redes bayesianas dinâmicas pode ser realizado como um subproduto de inferência. A inferência fornece uma estimativa de quais transições realmente ocorreram e de quais estados geraram as leituras de sensores, e essas estimativas podem ser usadas para atualizar os modelos. Os modelos atualizados fornecem novas estimativas, e o processo iterage para a convergência. O processo global é uma instância da maximização de expectativas, ou

algoritmo EM (veja a Seção 20.3).

Note que o aprendizado exige a inferência de suavização total, e não a filtragem, porque a suavização fornece melhores estimativas dos estados do processo. O aprendizado com filtragem pode deixar de convergir corretamente; por exemplo, considere o problema de aprender para solucionar assassinatos: a menos que você seja testemunha, a suavização será *sempre* exigida para se deduzir o que aconteceu na cena do crime, a partir das variáveis observáveis.

O restante desta seção descreve algoritmos genéricos para as quatro tarefas de inferência, independentemente do tipo particular de modelo empregado. Nas seções subsequentes serão descritas as melhorias específicas de cada modelo.

15.2.1 Filtragem e previsão

Como ressaltamos na Seção 7.7.3, um algoritmo de filtragem precisa manter uma estimativa do estado atual e atualizá-la, em vez de retroceder ao longo da história inteira de percepções para cada atualização (caso contrário, o custo de cada atualização aumenta à medida que o tempo passa). Em outras palavras, dado o resultado da filtragem até o tempo t, é possível calcular facilmente o resultado correspondente a $t + 1$ a partir da nova evidência $\mathbf{e}_{t+1}$. Ou seja,

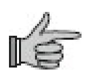
$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})) .$$

para alguma função f. Esse processo é chamado, com frequência, **avaliação recursiva**. Podemos visualizar o cálculo como se ele fosse de fato composta por duas partes: primeiro, a distribuição de estados atual é projetada adiante, de t para $t + 1$; em seguida, ela é atualizada com a utilização da nova evidência $\mathbf{e}_{t+1}$. Esse processo de duas partes emerge de forma bastante simples quando a fórmula é rearranjada:

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \quad \text{(repartindo a evidência)} \\ &= \alpha\, \mathbf{P}(\mathbf{e}_{1:t+1} \mid \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) \quad \text{(usando a regra de Bayes)} \\ &= \alpha\, \mathbf{P}(\mathbf{e}_{1:t+1} \mid \mathbf{X}_{t+1}) \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) \quad \text{(pela suposição de sensores de Markov).} \qquad (15.4)\end{aligned}$$

Aqui e ao longo deste capítulo, α é uma constante de normalização usada para fazer as probabilidades terem soma igual a 1. O segundo termo, $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t})$, representa uma previsão de um passo para o estado seguinte, e o primeiro termo atualiza esse outro com a nova evidência; note que $\mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1})$ pode ser obtida diretamente a partir do modelo de sensores. Agora, obtemos a previsão de um passo correspondente ao estado seguinte por condicionamento sobre o estado atual $\mathbf{X}_t$:

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) &= \alpha\, \mathbf{P}((\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \\ &= \alpha\, \mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \text{ (suposição de Markov).} \qquad (15.5)\end{aligned}$$

 Dentro do somatório, o primeiro fator é simplesmente o modelo de transição, e o segundo vem

da distribuição de estados atual. Consequentemente, temos a formulação recursiva desejada. Podemos imaginar a estimativa filtrada $\mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$ como uma "mensagem" $\mathbf{f}_{1:t}$ que é propagada ao longo da sequência, modificada por cada transição e atualizada por cada nova observação. O processo é dado por:

$$\mathbf{f}_{1:t+1} = \alpha \, \text{PARAFRENTE}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1})$$

onde PARAFRENTE implementa a atualização descrita na Equação 15.5 e o processo começa com $\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0)$. Quando todas as variáveis de estados são discretas, o tempo para cada atualização é constante (isto é, independentemente de t) e o espaço exigido também é constante. (É claro que as constantes dependem do tamanho do espaço de estados e do tipo específico do modelo temporal em questão.) *Os requisitos de tempo e de espaço para atualização devem ser constantes se um agente com memória limitada tiver de controlar a distribuição do estado atual sobre uma sequência ilimitada de observações*.

Vamos ilustrar o processo de filtragem para dois passos do exemplo básico do guarda-chuva (Figura 15.2). Isto é, vamos calcular $\mathbf{P}(R_2 \mid u_{1:2})$ como segue:

- No dia 0, não temos observações, apenas as crenças anteriores do guarda de segurança; vamos supor que consista em $\mathbf{P}(R0) = \langle 0{,}5, 0{,}5 \rangle$.
- No dia 1, o guarda-chuva aparece e, assim, $U_1 = \textit{verdadeiro}$. A previsão de $t = 0$ até $t = 1$ é dada por:

$$\mathbf{P}(R_1) = \sum_{r_0} \mathbf{P}(R_1|r_0)P(r_0)$$
$$= \langle 0{,}7, 0{,}3 \rangle \times 0{,}5 + \langle 0{,}3, 0{,}7 \rangle \times 0{,}5 = \langle 0{,}5, 0{,}5 \rangle.$$

  Em seguida, a etapa de atualização simplesmente multiplica pela probabilidade da evidência para $t = 1$ e normaliza, como mostrado na Equação 15.4:

$$\mathbf{P}(R_1 \mid u_1) = \alpha \, \mathbf{P}(u_1 \mid R_1)\mathbf{P}(R_1) = \alpha \langle 0{,}9, 0{,}2 \rangle \langle 0{,}5, 0{,}5 \rangle$$
$$= \alpha \langle 0{,}45, 0{,}1 \rangle \approx \langle 0{,}818, 0{,}182 \rangle.$$

- No dia 2, o guarda-chuva aparece, então $U_2 = \textit{verdadeiro}$. A previsão a partir de $t = 1$ até $t = 2$ é:

$$\mathbf{P}(R_2 \mid u_1) = \sum_{r_1} \mathbf{P}(R_2 \mid r_1)P(r_1 \mid u_1)$$
$$= \langle 0{,}7, 0{,}3 \rangle \times 0{,}818 + \langle 0{,}3, 0{,}7 \rangle \times 0{,}182 \approx \langle 0{,}627, 0{,}373 \rangle,$$

  e sua atualização com a evidência correspondente a $t = 2$ fornece:

$$\mathbf{P}(R_2 \mid u_1, u_2) = \alpha \, \mathbf{P}(u_2 \mid R_2)\mathbf{P}(R_2 \mid u_1) = \alpha \langle 0{,}9, 0{,}2 \rangle \langle 0{,}627, 0{,}373 \rangle$$
$$= \alpha \langle 0{,}565, 0{,}075 \rangle \approx \langle 0{,}883, 0{,}117 \rangle.$$

Intuitivamente, a probabilidade de chuva aumenta do dia 1 para o dia 2 porque a chuva persiste. O

Exercício 15.2(a) pede para investigar mais a fundo essa tendência.

A tarefa de **previsão** pode ser vista simplesmente como a filtragem sem a adição de nova evidência. De fato, o processo de filtragem já incorpora uma previsão de um passo e é fácil derivar o cálculo recursivo a seguir para prever o estado em $t + k + 1$ a partir de uma previsão para $t + k$:

$$\mathbf{P}(\mathbf{X}_{t+k+1} \mid \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \mathbf{P}(\mathbf{X}_{t+k+1} \mid \mathbf{x}_{t+k}) P(\mathbf{x}_{t+k} \mid \mathbf{e}_{1:t}) . \quad (15.6)$$

Naturalmente, esse cálculo envolve apenas o modelo de transição e não o modelo de sensores.

É interessante considerar o que acontece à medida que tentamos prever mais e mais à frente no futuro. Como mostra o Exercício 15.2(b), a distribuição prevista para chuva converge para um ponto fixo ⟨0,5, 0,5⟩, após o qual ela permanece constante durante todo o tempo. Essa é a **distribuição estacionária** do processo de Markov definido pelo modelo de transição. Conhecemos uma informação importante sobre as propriedades de tais distribuições e sobre o **tempo de mistura** — em linhas gerais, o tempo necessário para alcançar o ponto fixo. Em termos práticos, isso condena ao fracasso qualquer tentativa de prever o estado *real* para uma série de passos que seja maior que uma pequena fração do tempo de mistura, a menos que a distribuição estacionária em si esteja aumentada em uma pequena área do espaço de estados. Quanto mais incerteza existir no modelo de transição, mais curto será o tempo de mistura e mais o futuro ficará obscurecido.

Além da filtragem e da previsão, podemos usar uma recursão para a frente para calcular a **probabilidade** da sequência de evidência, $P(\mathbf{e}_{1:t})$. Essa é uma quantidade útil se quisermos comparar diferentes modelos temporais que podem ter produzido a mesma sequência de evidência (por exemplo, dois modelos diferentes para a persistência da chuva). No caso dessa recursão, utilizamos uma mensagem de probabilidade $l_{1:t}(\mathbf{X}_t) = \mathbf{P}(\mathbf{X}_t, \mathbf{e}_{1:t})$. É um exercício simples mostrar que o cálculo da mensagem é idêntico à mensagem da filtragem:

$$\ell_{1:t+1} = \text{PARAFRENTE}(\ell_{1:t}, \mathbf{e}_{t+1}) .$$

Tendo calculado $l_{1:t}$, obtemos a probabilidade real pelo somatório de $\mathbf{X}_t$:

$$L_{1:t} = P(\mathbf{e}_{1:t}) = \sum_{\mathbf{X}_t} \ell_{1:t}(\mathbf{x}_t). \quad (15.7)$$

Observe que a mensagem de probabilidade representa as probabilidades de sequências de evidências cada vez mais longas à medida que o tempo passa e, assim, se torna numericamente cada vez menor, levando a um problema de subfluxo com aritmética de ponto flutuante. Na prática esse é um problema importante, mas não vamos entrar no mérito das soluções aqui.

15.2.2 Suavização

Como vimos antes, a **suavização** é o processo de calcular a distribuição sobre estados anteriores, dada a evidência até o presente; isto é, $\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t})$ = para $0 \leq k < t$ (veja a Figura 15.3). Em

antecipação a outra abordagem de transmissão de mensagens recursiva, podemos dividir o cálculo em duas partes: a evidência até k e a evidência de $k + 1$ até t,

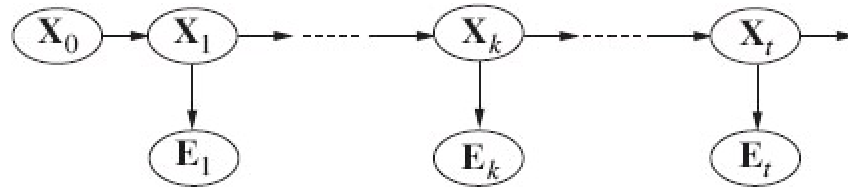


Figura 15.3 A suavização calcula $\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t})$, a distribuição posterior do estado em algum tempo passado k, dada uma sequência completa de observações desde 1 até t.

$$\begin{aligned}\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\ &= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{e}_{1:k}) \quad \text{(usando-se a regra de Bayes)} \\ &= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) \quad \text{(usando-se independência condicional)} \\ &= \alpha\, \mathbf{f}_{1:k} \times \mathbf{b}_{k+1:t}\,,\end{aligned} \qquad (15.8)$$

onde "×" representa multiplicação de vetores pontuais. Aqui definimos uma mensagem "para trás" $\mathbf{b}_{k+1:t} = \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k)$, análoga à mensagem para a frente $\mathbf{f}_{1:k}$. A mensagem para a frente $\mathbf{f}_{1:k}$ pode ser calculada pela filtragem para a frente de 1 até k, dada pela Equação 15.5. Em consequência disso, a mensagem para trás $\mathbf{b}_{k+1:t}$ pode ser calculada por um processo recursivo que funciona no *sentido inverso* a partir de t:

$$\begin{aligned}\mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad \text{(condicionamento sobre } X_{k+1}) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} \mid \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad \text{(por independência condicional)} \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} \mid \mathbf{x}_{k+1})P(\mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k)\,,\end{aligned} \qquad (15.9).$$

onde o último passo segue pela independência condicional de $\mathbf{e}_{k+1}$ e $\mathbf{e}_{k+2:t}$, dado $\mathbf{X}_{k+1}$. Dos três fatores desse somatório, o primeiro e o terceiro são obtidos diretamente a partir do modelo, e o segundo é a "chamada recursiva". Usando a notação de mensagem, temos:

$$\mathbf{b}_{k+1:t} = \text{PARATRÁS}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1:t})$$

onde PARATRÁS implementa a atualização descrita na Equação 15.9. Como ocorre no caso da recursão para a frente, o tempo e o espaço necessários para cada atualização são constantes e, desse modo, independentes de t. Agora, podemos ver que os dois termos da Equação 15.8 podem ser calculados por recursões através do tempo, um deles no sentido direto a partir de 1 até k e usando a equação de filtragem 15.5, e o outro funcionando no sentido inverso de t até $k + 1$ e usando a Equação 15.9. Observe que a fase para a frente é inicializada com $\mathbf{b}_{t+1:t} = \mathbf{P}(\mathbf{e}_{t+1:t} \mid \mathbf{X}_t) = \mathbf{P}(\ \mid \mathbf{X}_t\ )\mathbf{1}$, onde $\mathbf{1}$ é um vetor de valores unitários (como $\mathbf{e}_{t+1:t}$ é uma sequência vazia, a probabilidade de

observá-la é 1).

Agora, vamos aplicar esse algoritmo ao exemplo do guarda-chuva calculando a estimativa suavizada para a probabilidade de chuva em tempo k = 1, dadas as observações sobre o guarda-chuva nos dias 1 e 2. A partir da Equação 15.8, essa estim ativa é dada por:

$$\mathbf{P}(R_1 | u_1, u_2) = \alpha \mathbf{P}(R_1 | u_1)\mathbf{P}(u_2 | R_1) . \quad (15.10)$$

Já sabemos que o primeiro termo é ⟨0,818, 0,182⟩, a partir do processo de filtragem para a frente descrito anteriormente. O segundo termo pode ser calculado pela aplicação da recursão para trás na Equação 15.9:

$$\mathbf{P}(u_2 | R_1) = \sum_{r_2} P(u_2 | r_2)P(| r_2)\mathbf{P}(r_2 | R_1)$$
$$= (0{,}9 \times 1 \times \langle 0{,}7, 0{,}3\rangle) + (0{,}2 \times 1 \times \langle 0{,}3, 0{,}7\rangle) = \langle 0{,}69, 0{,}41\rangle.$$

Inserindo essa expressão na Equação 15.10, descobrimos que a estimativa suavizada para chuva no dia 1 é:

$$\mathbf{P}(R_1|u_1, u_2) = a \langle 0{,}818, 0{,}182\rangle \times \langle 0{,}69, 0{,}41\rangle \approx \langle 0{,}883, 0{,}117\rangle.$$

Desse modo, a estimativa suavizada de chuva no dia 1é *mais alta* que a estimativa filtrada (0,818), nesse caso. Isso ocorre porque o guarda-chuva no dia 2 torna mais provável ter chovido no dia 2; por sua vez, como a chuva tende a persistir, isso torna mais provável ter chovido no dia 1.

Ambas as recursões, para a frente e para trás, demoram um período de tempo constante por passo; consequentemente, a complexidade de tempo de suavização com relação à evidência $\mathbf{e}_{1:t}$ é $O(t)$. Essa é a complexidade para suavização em um período de tempo específico k. Se quisermos suavizar a sequência inteira, um método óbvio será simplesmente executar todo o processo de suavização, uma vez para cada período de tempo a ser suavizado. Isso resulta em uma complexidade de tempo igual a $O(t^2)$. Uma abordagem melhor utiliza uma aplicação muito simples de programação dinâmica para reduzir a complexidade a $O(t)$. Aparece uma pista na análise precedente do exemplo de guarda-chuva, onde fomos capazes de reutilizar os resultados da fase de filtragem para a frente. A chave para o algoritmo de tempo linear é *registrar os resultados* da filtragem para a frente sobre a sequência inteira. Em seguida, executamos a recursão para trás desde t até 1, calculando a estimativa suavizada em cada passo k da mensagem para trás calculada $\mathbf{b}_{k+1:t}$ e da mensagem para a frente armazenada $\mathbf{f}_{1:k}$. O algoritmo, apropriadamente chamado **algoritmo para a frente-para trás**, é mostrado na Figura 15.4.

função PARAFRENTE-PARATRÁS(**ev**, *anterior*) **retorna** um vetor de distribuições de probabilidade
entradas: ev, um vetor de valores de evidência correspondentes aos passos 1, . . . , t
anterior, a distribuição anterior sobre o estado inicial, $\mathbf{P}(\mathbf{X}_0)$
variáveis locais: fv, um vetor de mensagens para a frente correspondentes aos passos 0, . . . , t
b, uma representação da mensagem para trás, formada inicialmente apenas

por valores 1
sv, um vetor de estimativas suavizadas correspondentes aos passos 1, ... , t

fv[0] ← *anterior*
para i = 1 **até** t **faça**
fv[i] ← PARAFRENTE(**fv**[i – 1], **ev**[i])
para i = t **descendo até** 1 **faça**
sv[i] ← NORMALIZAR(**fv**[i] × **b**)
b ← PARATRÁS(**b**, **ev**[i])
retornar sv

Figura 15.4 Algoritmo para a frente-para trás de suavização para cálculo de probabilidades posteriores de uma sequência de estados, dada uma sequência de observações. Os operadores PARAFRENTE e PARATRÁS são definidos pelas Equações 15.5 e 15.9, respectivamente.

O leitor atento terá notado que a estrutura de rede bayesiana da Figura 15.3 é uma *poliárvore*, como definido anteriormente. Isso significa que uma aplicação para a frente do algoritmo de formação de agrupamentos também resulta em um algoritmo de tempo linear que calcula estimativas suavizadas para a sequência inteira. Agora, demonstramos que o algoritmo para a frente-para trás é na realidade um caso especial do algoritmo de propagação de poliárvore utilizado com métodos de formação de agrupamentos (embora os dois tenham sido desenvolvidos de modo independente).

O algoritmo para a frente-para trás forma a espinha dorsal computacional que lida com sequências de observações com ruído, variando desde o reconhecimento da fala até o acompanhamento de aeronaves por radar. Conforme descrevemos, ele apresenta duas desvantagens de ordem prática. A primeira é que sua complexidade de espaço pode ser muito alta para aplicações nas quais o espaço de estados é grande e as sequências são longas. O algoritmo utiliza o espaço $O(|\mathbf{f}|t)$, onde $|\mathbf{f}|$ é o tamanho da representação da mensagem para a frente. O requisito espacial pode ser reduzido para $O(|\mathbf{f}| \log t)$, com aumento concomitante na complexidade de tempo por um fator igual a $\log t$, como mostra o Exercício 15.3. Em alguns casos (veja a Seção 15.3), um algoritmo de espaço constante pode ser usado sem qualquer penalidade de tempo.

A segunda desvantagem do algoritmo básico é que ele precisa ser modificado para funcionar em uma configuração *on-line* na qual as estimativas suavizadas devem ser calculadas para fatias de tempo anteriores, à medida que novas observações são continuamente adicionadas ao final da sequência. O requisito mais comum é o de **suavização de retardo fixo**, que exige o cálculo da estimativa suavizada $\mathbf{P}(\mathbf{X}_{t-d} \mid \mathbf{e}_{1:t})$ para d fixo. Isto é, a suavização é feita para a fatia de tempo d passos atrasada em relação ao tempo atual t; conforme t aumenta, a suavização tem de continuar. É óbvio que podemos executar o algoritmo para a frente-para trás sobre a "janela" de d passos à medida que cada nova observação é adicionada, mas isso parece ineficiente. Na Seção 15.3, veremos que a suavização de retardo fixo pode, em alguns casos, ser feita em tempo constante por atualização, independentemente do retardo d.

15.2.3 Como descobrir a sequência mais provável

Suponha que [*verdadeiro*, *verdadeiro*, *falso*, *verdadeiro*, *verdadeiro*] seja a sequência de guarda-chuva para os cinco primeiros dias de serviço do guarda de segurança. Qual é a sequência de condições do clima com maior probabilidade de explicar isso? A ausência do guarda-chuva no dia 3 significa que não estava chovendo ou será que o diretor se esqueceu de trazer o guarda-chuva? Se não tivesse chovido no dia 3, talvez (porque as condições do clima tendem a persistir) também não tivesse chovido no dia 4, mas o diretor pode ter trazido o guarda-chuva só por precaução. Ao todo, existem 2^5 possíveis sequências de condições do clima que poderíamos selecionar. Existe um meio de encontrar a mais provável que não seja enumerando todas elas?

Poderíamos tentar o procedimento de tempo linear a seguir: usar o algoritmo de suavização para descobrir a distribuição posterior para as condições do clima em cada período; em seguida, construir a sequência, utilizando em cada passo as condições do clima mais provavelmente concordantes com a distribuição posterior. Tal abordagem deve fazer soar um alarme na cabeça do leitor porque as distribuições posteriores calculadas pela suavização são distribuições sobre períodos de tempo *isolados*; por outro lado, para encontrar a *sequência* mais provável temos de considerar probabilidades *conjuntas* sobre todos os períodos de tempo. Os resultados podem de fato ser bastante diferentes (veja o Exercício 15.4).

Existe um algoritmo de tempo linear para encontrar a sequência mais provável, mas ele exige um pouco mais de reflexão. Esse algoritmo se baseia na mesma propriedade de Markov que gerou algoritmos eficientes para filtragem e suavização. O modo mais fácil de pensar no problema é visualizar cada sequência como um *caminho* por um grafo cujos nós são os *estados* possíveis em cada período de tempo. Mostramos esse grafo para o mundo do guarda-chuva na Figura 15.5(a). Agora, considere a tarefa de descobrir o caminho mais provável por esse grafo, onde a probabilidade de qualquer caminho é o produto das probabilidades de transição ao longo do caminho pelas probabilidades das observações dadas em cada estado. Vamos nos concentrar em particular nos caminhos que alcançam o estado $Chuva_5 = verdadeiro$. Devido à propriedade de Markov, segue-se que o caminho mais provável para o estado $Chuva_5 = verdadeiro$ consiste no caminho mais provável até *algum* estado no tempo 4, seguido por uma transição para $Chuva_5 = verdadeiro$; e o estado no tempo 4 que se tornará parte do caminho para $Chuva_5 = verdadeiro$ será aquele que maximizar a probabilidade desse caminho. Em outras palavras, *existe um relacionamento recursivo entre caminhos mais prováveis até cada estado* $\mathbf{x}_{t+1}$ *e caminhos mais prováveis até cada estado* $\mathbf{x}_t$. Podemos escrever esse relacionamento como uma equação que conecta as probabilidades dos caminhos:

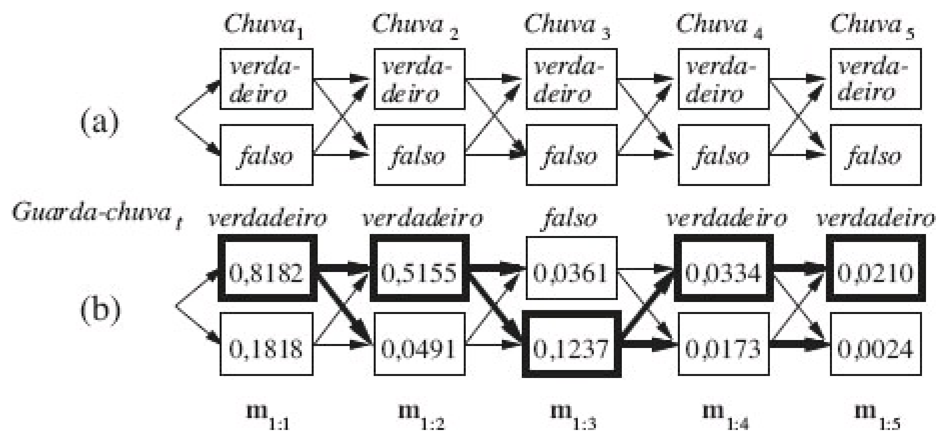


Figura 15.5 (a) Sequências de estados possíveis para $Chuva_t$ podem ser visualizadas como caminhos através de um grafo dos estados possíveis em cada período de tempo (os estados são mostrados como nós quadrados para evitar confusão com os nós de uma rede bayesiana). (b) Operação do algoritmo de Viterbi para a sequência de observação de guarda-chuva [*verdadeiro*, *verdadeiro*, *falso*, *verdadeiro*, *verdadeiro*]. Para cada período de tempo t, mostramos os valores da mensagem $\mathbf{m}_{1:t}$, que fornece a probabilidade de a melhor sequência alcançar cada estado no tempo t. Além disso, para cada estado, a seta grossa que leva até ele indica seu melhor predecessor. Seguindo as setas grossas de volta a partir do estado mais provável em **m**1:5, temos a sequência mais provável.

$$\begin{aligned}
&\max_{\mathbf{x}_1\ldots\mathbf{x}_t} \mathbf{P}(\mathbf{x}_1,\ldots,\mathbf{x}_t,\mathbf{X}_{t+1}\,|\,\mathbf{e}_{1:t+1}) \\
&= \alpha\,\mathbf{P}(\mathbf{e}_{t+1}\,|\,\mathbf{X}_{t+1}) \max_{\mathbf{x}_t}\left(\mathbf{P}(\mathbf{X}_{t+1}\,|\,\mathbf{x}_t) \max_{\mathbf{x}_1\ldots\mathbf{x}_{t-1}} P(\mathbf{x}_1,\ldots,\mathbf{x}_{t-1},\mathbf{x}_t\,|\,\mathbf{e}_{1:t})\right) .
\end{aligned} \tag{15.11}$$

A Equação 15.11 é *idêntica* à equação de filtragem 15.5, exceto pelo fato de que:

1. A mensagem para a frente $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t\,|\,\mathbf{e}_{1:t})$ é substituída pela mensagem:

$$\mathbf{m}_{1:t} = \max_{\mathbf{x}_1\ldots\mathbf{x}_{t-1}} \mathrm{P}(\mathbf{x}_1,\ldots,\mathbf{x}_{t-1},\mathbf{X}_t\,|\,\mathbf{e}_{1:t}),$$

   isto é, as probabilidades do caminho mais provável até cada estado $\mathbf{x}_t$; e

2. O somatório sobre $\mathbf{x}_t$ na Equação 15.5 é substituído pela maximização sobre $\mathbf{x}_t$ na Equação 15.11.

Desse modo, o algoritmo para calcular a sequência mais provável é semelhante à filtragem: ele percorre a sequência no sentido para a frente calculando a mensagem **m** em cada período de tempo utilizando a Equação 15.11. O progresso desse cálculo é mostrado na Figura 15.5(b). No final, ele terá a probabilidade para a sequência mais provável que alcança *cada um* dos estados finais. Pode-se, portanto, selecionar com facilidade a sequência mais provável de todas (o estado com contorno em negrito). Com a finalidade de identificar a sequência real, em vez de simplesmente calcular sua probabilidade, o algoritmo também precisará registrar, para cada estado, o melhor estado que conduz a isso, o que está indicado pelas setas em negrito na Figura 15.5(b). A sequência ótima é identificada seguindo-se os ponteiros de volta a partir do melhor estado final.

O algoritmo que acabamos de descrever denomina-se **algoritmo de Viterbi**, em homenagem a seu criador. Como o algoritmo de filtragem, sua complexidade é linear em t, a duração da sequência.

Porém, diferentemente da filtragem, que utiliza espaço constante, seu requisito de espaço também é linear em t. Isso ocorre porque o algoritmo de Viterbi precisa manter os ponteiros que identificam a melhor sequência que leva a cada estado.

15.3 MODELOS OCULTOS DE MARKOV

A seção precedente desenvolveu algoritmos para raciocínio probabilístico temporal usando uma estrutura geral independente da forma específica dos modelos de transição e de sensores. Nesta e nas duas seções seguintes, discutiremos modelos mais concretos e aplicações que ilustram a capacidade dos algoritmos básicos e, em alguns casos, permitem aperfeiçoamentos adicionais.

Começaremos com o **modelo oculto de Markov**, ou **MOM**. Um MOM é um modelo probabilístico temporal no qual o estado do processo é descrito por uma *única* variável aleatória *discreta*. Os valores possíveis da variável são os estados possíveis do mundo. O exemplo do guarda-chuva descrito na seção precedente é então um MOM, pois ele tem apenas uma variável de estado: $Chuva_t$. O que acontece se você tiver um modelo com duas ou mais variáveis de estado? Você pode ainda se encaixar no quadro do MOM combinando todas as variáveis de estados em uma única "megavariável", cujos valores são todas as tuplas de valores possíveis das variáveis de estados individuais. Veremos que a estrutura restrita de MOMs permite a implementação de uma matriz muito simples e elegante de todos os algoritmos básicos.[4]

15.3.1 Algoritmos matriciais simplificados

Com uma única variável de estado discreta X_t, podemos dar forma concreta às representações do modelo de transição, do modelo de sensores e das mensagens para a frente e para trás. Seja a variável de estado X_t com valores denotados por inteiros 1, …, S, onde S é o número de estados possíveis. O modelo de transição $\mathbf{P}(X_t \mid X_{t-1})$ se torna uma matriz $\mathbf{T}$ $S \times S$, onde:

$$\mathbf{T}_{ij} = P(X_{t=j} \mid X_{t-1=i}).$$

Ou seja. $\mathbf{T}_{ij}$ é a probabilidade de uma transição do estado i para o estado j. Por exemplo, a matriz de transição para o mundo de guarda-chuva é:

$$\mathbf{T} = \mathbf{P}(X_t \mid X_{t-1}) = \begin{pmatrix} 0{,}7 & 0{,}3 \\ 0{,}3 & 0{,}7 \end{pmatrix}.$$

Também colocamos o modelo de sensores em forma de matriz. Nesse caso, como o valor da variável de evidência E_t é conhecido no tempo t (chamemos e_t), precisamos apenas especificar, para cada estado, com qual probabilidade o estado faz com que e_t apareça: precisamos de $P(e_t \mid X_{t=i})$ para cada estado i. Para conveniência matemática colocamos esses valores em uma matriz diagonal $S \times S$ $\mathbf{O}_t$ cuja iésima entrada diagonal é $P(e_t \mid X_{t=i})$ e cujas outras entradas são 0. Por exemplo, no dia

1, no mundo do guarda-chuva da Figura 15.5, U_1 = *verdadeiro* e, no dia 3, U_3 = *falso*; assim, da Figura 15.2, temos:

$$\mathbf{O}_1 = \begin{pmatrix} 0{,}9 & 0 \\ 0 & 0{,}2 \end{pmatrix}; \quad \mathbf{O}_3 = \begin{pmatrix} 0{,}1 & 0 \\ 0 & 0{,}8 \end{pmatrix}.$$

Agora, se utilizarmos vetores coluna para representar as mensagens para a frente e para trás, as computações se tornarão simples operações de vetores de matrizes. A equação para a frente (15.5) se torna:

$$\mathbf{f}_{1:t+1} = \alpha\, \mathbf{O}_{t+1} \mathbf{T}^{\top} \mathbf{f}_{1:t} \tag{15.12}$$

e a equação para trás (15.9) se torna:

$$\mathbf{b}_{k+1:t} = \mathbf{T}\mathbf{O}_{k+1} \mathbf{b}_{k+2:t}. \tag{15.13}$$

A partir dessas equações, podemos ver que a complexidade de tempo do algoritmo para a frente-para trás (Figura 15.4) aplicada a uma sequência cuja duração t é $O(S^2 t)$ porque cada passo exige a multiplicação de um vetor de S elementos por uma matriz $S \times S$. O requisito de espaço é $O(St)$ porque a passagem para a frente armazena t vetores de tamanho S.

Além de fornecer uma descrição elegante dos algoritmos de filtragem e suavização para MOMs, a formulação de matriz revela oportunidades para algoritmos otimizados. O primeiro é uma simples variação sobre o algoritmo para a frente-para trás que permite que a suavização seja executada em espaço *constante*, independentemente da duração da sequência. A ideia é que a suavização para qualquer fatia de tempo k específica exige a presença simultânea das mensagens para a frente e para trás, $\mathbf{f}_{1:k}$ e $\mathbf{b}_{k+1:t}$, de acordo com a Equação 15.8. O algoritmo para a frente-para trás consegue isso armazenando os valores **f** calculados na passagem para a frente, de modo que eles fiquem disponíveis durante a passagem para trás. Outro modo de conseguir isso é usar uma única passagem que efetue a propagação de **f** e de **b**, ambas no mesmo sentido. Por exemplo, a mensagem “para a frente” **f** pode ser propagada no sentido para trás se manipularmos a Equação 15.12 para atuar no outro sentido:

$$\mathbf{f}_{1:t} = \alpha' (\mathbf{T}^{\top})^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1}.$$

O algoritmo de suavização modificado funciona executando primeiro a passagem para a frente padrão para calcular $\mathbf{f}_{t:t}$ (ignorando-se todos os resultados intermediários) e depois executando a passagem para trás para **b** e **f** juntas, utilizando-se essas mensagens para calcular a estimativa suavizada em cada passo. Tendo em vista que é necessária apenas uma cópia de cada mensagem, os requisitos de armazenamento são constantes (isto é, independentes de t, a duração da sequência). Existem duas restrições significativas sobre esse algoritmo: ele exige que a matriz de transição tenha matriz inversa e que o modelo de sensores não tenha zeros, isto é, que toda observação seja possível em todo estado.

Uma segunda área em que a formulação de matriz revela um aperfeiçoamento é a suavização on-line com retardo fixo. O fato de ser possível realizar a suavização em espaço constante sugere que

deve existir um algoritmo recursivo eficiente para suavização on-line, isto é, um algoritmo cuja complexidade de tempo seja independente da duração do retardo. Vamos supor que o retardo seja d; isto é, estamos realizando a suavização na fatia de tempo $t - d$, onde o tempo atual é t. Pela Equação 15.8, precisamos calcular:

$$\alpha \mathbf{f}_{1:t-d} \times \mathbf{b}_{t-d+1:t}$$

para a fatia $t - d$. Então, quando chegar uma nova observação, precisaremos calcular:

$$\alpha \mathbf{f}_{1:t-d+1} \times \mathbf{b}_{t-d+2:t+1}$$

para a fatia $t - d + 1$. Como isso pode ser feito de modo incremental? Primeiro, podemos calcular $\mathbf{f}_{1:t-d+1}$ a partir de $\mathbf{f}_{1:t-d}$ utilizando o processo de filtragem padrão, segundo a Equação 15.5.

Calcular a mensagem para trás de modo incremental é uma ação mais complicada porque não existe nenhum relacionamento simples entre a mensagem para trás antiga $\mathbf{b}_{t-d+1:t}$ e a nova mensagem para trás $\mathbf{b}_{t-d+2:t+1}$. Em vez disso, examinaremos o relacionamento entre a mensagem para trás antiga $\mathbf{b}_{t-d+1:t}$ e a mensagem para trás no início da sequência, $\mathbf{b}_{t+1:t}$. Para isso, aplicamos a Equação 15.13 d vezes para obter:

$$\mathbf{b}_{t-d+1:t} = \left(\prod_{i=t-d+1}^{t} \mathbf{TO}_i \right) \mathbf{b}_{t+1:t} = \mathbf{B}_{t-d+1:t}\mathbf{1} , \qquad (15.14)$$

onde a matriz $\mathbf{B}_{t-d+1:t}$ é o produto da sequência de matrizes $\mathbf{T}$ e $\mathbf{O}$. $\mathbf{B}$ pode ser considerado um "operador de transformação", que transforma uma mensagem para trás posterior em uma anterior. Uma equação semelhante é válida para as novas mensagens para trás *depois* da chegada da observação seguinte:

$$\mathbf{b}_{t-d+2:t+1} = \left(\prod_{i=t-d+2}^{t+1} \mathbf{TO}_i \right) \mathbf{b}_{t+2:t+1} = \mathbf{B}_{t-d+2:t+1}\mathbf{1} . \qquad (15.15)$$

Examinando as expressões de produtos nas Equações 15.14 e 15.15, vemos que elas têm um relacionamento simples: para obter o segundo produto, "divide-se" o primeiro produto pelo primeiro elemento $\mathbf{TO}_{t-d+1}$ e multiplica-se pelo último elemento novo $\mathbf{TO}_{t+1}$. Então, em linguagem de matrizes, existe um relacionamento simples entre as matrizes $\mathbf{B}$ antiga e nova:

$$\mathbf{B}_{t-d+2:t+1} = \mathbf{O}_{t-d+1}^{-1}\mathbf{T}^{-1}\mathbf{B}_{t-d+1:t}\mathbf{TO}_{t+1} . \qquad (15.16)$$

Essa equação fornece uma atualização incremental para a matriz $\mathbf{B}$ que, por sua vez (pela Equação 15.15), nos oferece a possibilidade de calcular a nova mensagem para trás $\mathbf{b}_{t-d+2:t+1}$. O algoritmo completo, que exige armazenamento e atualização de $\mathbf{f}$ e $\mathbf{B}$, é mostrado na Figura 15.6.

função SUAVIZAÇÃO-DE-RETARDO-FIXO(*et*, *mom*, *d*) **retorna** uma distribuição sobre $\mathbf{X}_{t-d}$
  entradas: *et*, a evidência atual por período de tempo *t*

mom, um modelo oculto de Markov com matriz de transição **T** $S \times S$
d, a duração do retardo para suavização

persistente: *t*, o tempo atual, inicialmente 1
f, a mensagem para a frente $\mathbf{P}(X_t \mid e_{1:t})$, inicialmente ANTERIOR[*mom*]
B, a matriz de transformação de *d* passos para trás, inicialmente a matriz identidade
$e_{t-d:t}$, lista dupla de evidências a partir de $t - d$ para *t*, inicialmente vazia

variáveis locais: $\mathbf{O}_{t-d}$, $\mathbf{O}_t$, matrizes diagonais contendo as informações do modelo de sensores

somar *et* ao final de $e_{t-d:t}$
$\mathbf{O}_t \leftarrow$ matriz diagonal contendo $\mathbf{P}(e_t | X_t)$
se $t > d$ **então**
  $\mathbf{f} \leftarrow$ PARAFRENTE(**f**, e_t)
  remover e_{t-d} do início de $e_{t-d:t}$
  $\mathbf{O}_{t-d} \leftarrow$ matriz diagonal contendo $\mathbf{P}(e_{t-d} \mid X_{t-d})$
  $\mathbf{B} \leftarrow \mathbf{O}^{-1}_{t-d}\ \mathbf{T}^{-1}\ \mathbf{BTO}_t$
senão $\mathbf{B} \leftarrow \mathbf{BTO}t$
$t \leftarrow t + 1$
se $t > d$ **então retornar** NORMALIZAR($\mathbf{f} \times \mathbf{B1}$) **senão retornar** nulo

Figura 15.6 Um algoritmo para suavização com retardo de tempo fixo de *d* passos, implementado como um algoritmo on-line que mostra como saída a nova estimativa suavizada, dada a observação correspondente a um novo período de tempo. Observe que o produto final NORMALIZAR($\mathbf{f} \times \mathbf{B1}$) é apenas *a* $\mathbf{f} \times \mathbf{b}$, pela Equação 15.14.

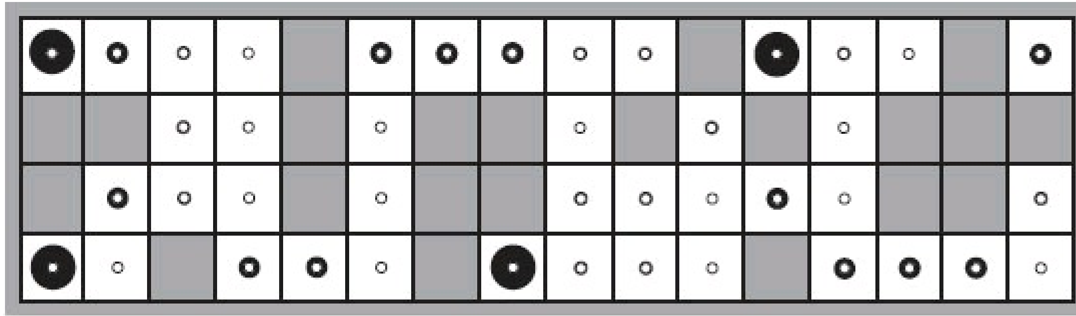
15.3.2 Exemplo do modelo oculto de Markov: localização

Foi introduzida uma forma simples de problema de **localização** para o mundo do aspirador de pó. Nessa versão, o robô teve uma única ação de *Movimento* não determinístico e seus sensores relataram perfeitamente se havia ou não obstáculos localizados ao norte, sul, leste e oeste; o estado de crença do robô era o conjunto de locais possíveis onde ele poderia estar.

Aqui nós tornamos o problema um pouco mais realista incluindo um modelo de probabilidade simples para o movimento do robô e permitindo o ruído nos sensores. O estado variável X_t representa a localização do robô na grade discreta; o domínio dessa variável é o conjunto de quadrados vazios $\{s_1, \ldots, s_n\}$. Seja VIZINHOS(*s*) o conjunto de quadrados vazios que são adjacentes a *s* e seja *N*(*s*) o tamanho desse conjunto. Então, o modelo de transição para a ação *Mover* diz que o robô tem a mesma probabilidade de acabar em qualquer quadrado vizinho:

$$P(X_{t+1} = j \mid X_t = i) = \mathbf{T}_{ij} = (1/N(i) \text{ se } j \in \text{VIZINHOS}(i) \text{ senão } 0).$$

Não sabemos onde o robô começa, então vamos supor uma distribuição uniforme sobre todos os quadrados, isto é, $P(X_0 = i) = 1/n$. Para o ambiente especial que consideramos (Figura 15.7), $n = 42$ e a matriz de transição **T** tem $42 \times 42 = 1.764$ entradas.


(a) Distribuição posterior sobre a localização do robô após $E_1 =$ NSW

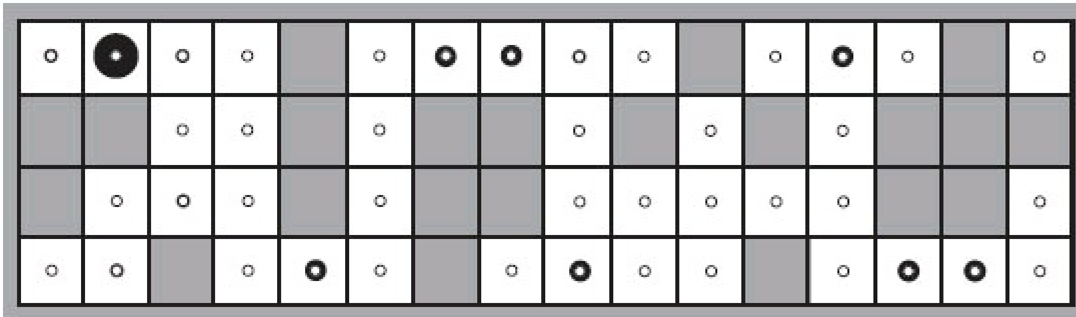

(b) Distribuição posterior sobre a localização do robô após $E_1 =$ NSW, $E_2 =$ NS

Figura 15.7 Distribuição posterior sobre a localização do robô: (a) uma observação $E_1 = NSO$; (b) depois de uma segunda observação $E_2 = NS$. O tamanho de cada disco corresponde à probabilidade de que o robô esteja naquela localização. A taxa de erro do sensor é $\in = 0{,}2$.

O sensor variável E_t tem 16 valores possíveis, cada sequência de quatro bits informando a presença ou ausência de um obstáculo em uma direção particular. Utilizaremos a notação *NS*, por exemplo, para significar que os sensores do norte e do sul relatam um obstáculo enquanto os do leste e oeste não relatam. Suponha que cada taxa de erro do sensor seja $\in$ e os erros ocorrem independentemente das quatro direções do sensor. Nesse caso, a probabilidade de obter todos os quatro bits da direita é $(1 - \in)^4$ e a probabilidade de obter todos errados é $\in^4$. Além disso, se d_{it} é a discrepância — o número de bits que são diferentes — entre os valores verdadeiros do quadrado i e a leitura real de e_t, então a probabilidade de que um robô no quadrado i vá receber uma leitura do sensor e_t é

$$P(E_t = e_t \mid X_t = i) = \mathbf{O}_{t_{ii}} = (1 - \in)^{4-d_{it}} \in^{d_{it}}.$$

Por exemplo, a probabilidade de que um quadrado com obstáculos para o norte e para o sul produza um sensor de leitura *NSL* é $(1 - \in)^3 \in^1$.

Dadas as matrizes **T** e $\mathbf{O}_t$, o robô pode usar a Equação 15.12 para calcular a distribuição posterior sobre as localizações, isto é, resolver onde ele está. A Figura 15.7 mostra as distribuições $\mathbf{P}(X_1 \mid E_1 = NSO)$ e $\mathbf{P}(X_2 \mid E_1 = NSO, E_2 = NS)$. Esse é o mesmo labirinto que vimos antes na Figura 4.18, mas lá usamos filtragem lógica para encontrar as localizações *possíveis*, assumindo sensoriamento

perfeito. Essas mesmas localizações ainda são as mais prováveis com sensoriamento ruidoso, mas agora *cada* localização tem uma probabilidade diferente de zero.

Além da filtragem para estimar sua localização atual, o robô pode usar suavização (Equação 15.13) para perceber onde estava em qualquer tempo passado — por exemplo, onde começou no tempo 0 — e pode usar o algoritmo de Viterbi para calcular o caminho mais provável que tenha tomado para chegar onde está agora. A Figura 15.8 mostra o erro de localização e a precisão do caminho de Viterbi para vários valores da taxa de erro do sensor per-bit. Mesmo quando ∈ for 20% — o que significa que a leitura do sensor está errada 59% do tempo total — geralmente o robô é capaz de calcular sua localização dentro de dois quadrados depois de 25 observações. Isso por causa da habilidade do algoritmo de integrar a evidência ao longo do tempo e levar em conta as restrições probabilísticas impostas na sequência de localização pelo modelo de transição. Quando ∈ for 10%, será difícil distinguir o desempenho após meia dúzia de observações do desempenho com sensoriamento perfeito. O Exercício 15.7 pede que você explore a robustez do algoritmo de localização de MOM em relação a erros na distribuição anterior $\mathbf{P}(X_0)$ e no modelo de transição em si. De modo geral, os níveis elevados de localização e de precisão de caminho são mantidos mesmo em face de erros substanciais nos modelos utilizados.

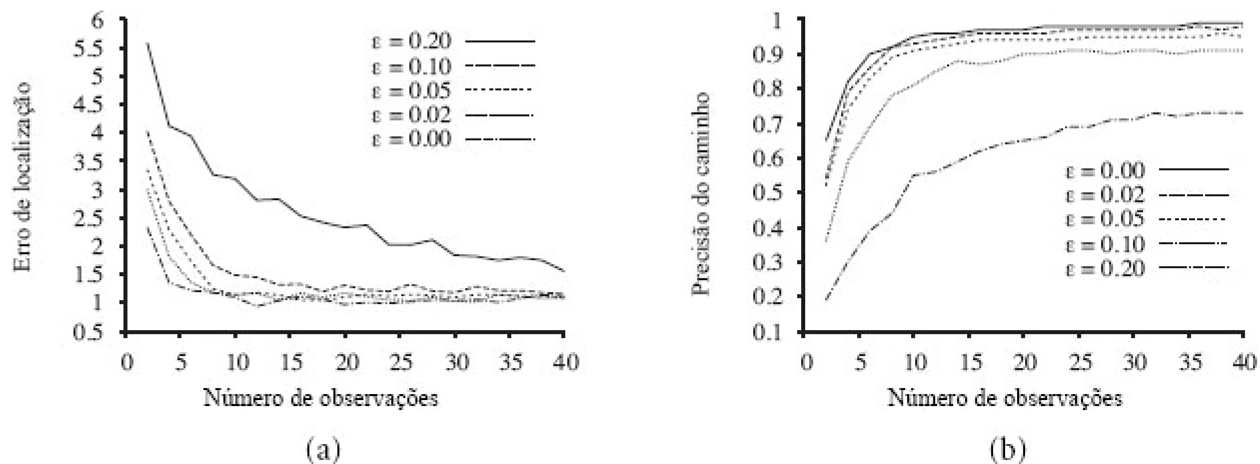


Figura 15.8 Desempenho da localização do MOM como função do comprimento da sequência de observação de vários valores diferentes de probabilidade de erro do sensor; dados calculados sobre 400 execuções. (a) Erro de localização, definido como a distância de Manhattan da localização verdadeira. (b) Precisão do caminho de Viterbi, definida como fração de estados corretos sobre o caminho de Viterbi.

A variável de estado para o exemplo que consideramos nesta seção é a localização física no mundo. Outros problemas podem, é claro, incluir outros aspectos do mundo. O Exercício 15.8 pede que você considere uma versão do robô do aspirador de pó cujo programa de ação seja ir em linha reta enquanto pode; só quando encontra um obstáculo, ele muda para uma nova direção (selecionada aleatoriamente). Para modelar esse robô, cada estado no modelo consiste em um par (*localização, direção*). Para o ambiente na Figura 15.7, que tem 42 quadrados vazios, isso leva a 168 estados e uma matriz de transição com 168^2 = 28. 224 entradas — um número ainda administrável. Se adicionarmos a possibilidade de sujeira nos quadrados, o número de estados será multiplicado por 2^{42} e a matriz de transição terminará com mais de 10^{29} entradas — não mais um número gerenciável; a Seção 15.5 mostra como usar redes bayesianas dinâmicas para modelar domínios com muitas

variáveis de estado. Se permitirmos que o robô se mova continuamente em vez de uma grade discreta, o número de estados torna-se infinito; a próxima seção mostra como lidar com esse caso.

15.4 FILTROS DE KALMAN

Imagine assistir a um pequeno pássaro voando entre a folhagem da selva densa ao entardecer: você vislumbra *flashes* breves e intermitentes de movimento, faz todo o possível para adivinhar onde o pássaro está e onde ele aparecerá em seguida, para não perdê-lo de vista. Ou, ainda, imagine que você seja um operador de radar da Segunda Guerra Mundial que perscruta um blipe débil e errante que surge uma vez a cada 10 segundos na tela. Ou, então, indo um pouco mais longe, imagine que você seja Kepler tentando reconstruir os movimentos dos planetas a partir de uma coleção de observações angulares altamente inexatas, tomadas a intervalos irregulares e medidos de forma imprecisa. Em todos esses casos, você está fazendo a filtragem: estimando variáveis de estado (aqui, posição e velocidade) a partir de observações com ruído. Se as variáveis fossem discretas, poderíamos modelar o sistema com o modelo oculto de Markov. Esta seção examina métodos de lidar com variáveis contínuas usando um algoritmo chamado **filtragem de Kalman**, em homenagem ao seu criador, Rudolf E. Kalman.

O voo do pássaro poderia ser especificado pela posição (X_t, Y_t, Z_t) e pela velocidade ($\dot{X}_t, \dot{Y}_t, \dot{Z}_t$) em cada instante no tempo. Também precisaremos de densidades condicionais adequadas para representar os modelos de transição e de sensores; como no Capítulo 14, utilizaremos distribuições **gaussianas lineares**. Isso significa que o próximo estado $\mathbf{X}_{t+1}$ deve ser uma função linear do estado atual $\mathbf{X}_t$ somada a algum ruído gaussiano, uma condição que acaba por ser bastante razoável na prática. Por exemplo, considere a coordenada X do pássaro, ignorando por enquanto as outras coordenadas. Seja Δ o intervalo de tempo entre observações e suponha velocidade constante; então, a atualização de posição é dada por $X_{t+\Delta} = X_t + \dot{X}\,\Delta$. Se adicionarmos ruído gaussiano (levando em conta a variação do vento etc.), teremos um modelo de transição gaussiano linear:

$$P(X_{t+\Delta} = x_{t+\Delta} \mid X_t = x_t, \dot{X}_t = \dot{x}_t) = N(x_t + \dot{x}_t\,\Delta, \sigma^2)(x_{t+\Delta}) \,.$$

A estrutura de rede bayesiana para um sistema com posição vetor $\mathbf{X}_t$ e velocidade $\dot{\mathbf{X}}_t$ é mostrada na Figura 15.9. Observe que essa é uma forma muito específica do modelo gaussiano linear; a forma geral será descrita mais adiante nesta seção e cobrirá uma vasta série de aplicações, além dos exemplos de movimento simples do primeiro parágrafo. O leitor talvez deseje consultar no Apêndice A algumas propriedades matemáticas de distribuições gaussianas; para nossos propósitos imediatos, a mais importante é que uma distribuição **gaussiana multivariada** para d variáveis é especificada por uma média μ de d elementos e uma matriz de covariância Σ $d \times d$.

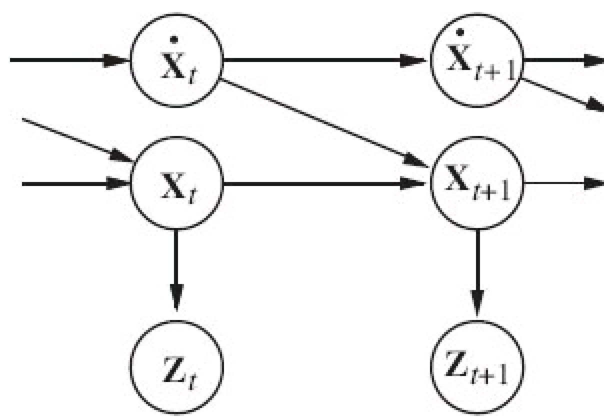


Figura 15.9 Estrutura de rede bayesiana para um sistema linear dinâmico com posição $\mathbf{X}_t$, velocidade $\dot{\mathbf{X}}_t$ e medida de posição $\mathbf{Z}t$.

15.4.1 Atualização de distribuições gaussianas

No Capítulo 14 aludimos a uma propriedade-chave da família de distribuições gaussianas lineares: ela permanece fechada sob as operações-padrão de redes bayesianas. Aqui, tornamos essa asserção precisa no contexto de filtragem em um modelo de probabilidade temporal. As propriedades exigidas correspondem ao cálculo de filtragem de dois passos na Equação 15.5:

1. Se a distribuição atual $\mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$ é gaussiana e o modelo de transição $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t)$ é gaussiana linear, então a distribuição prevista de um passo definido por

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) = \int_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t})\, d\mathbf{x}_t \qquad (15.17)$$

também é uma distribuição gaussiana.

2. Se a previsão $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ é gaussiana e o modelo de sensores $\mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1})$ é gaussiano linear, depois do condicionamento sobre a nova evidência, a distribuição atualizada:

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) = \alpha\, \mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1})\, \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) \qquad (15.18)$$

também é uma distribuição gaussiana.

Desse modo, o operador PARAFRENTE para filtragem de Kalman recebe uma mensagem gaussiana para a frente $\mathbf{f}_{1:t}$, especificada por uma média $\boldsymbol{\mu}_t$ e uma matriz de covariância Σ_t, e produz uma nova mensagem gaussiana para a frente multivariada $\mathbf{f}_{1:t+1}$, especificada por uma média $\boldsymbol{\mu}_{t+1}$ e uma matriz de covariância Σ_{t+1}. Assim, se começarmos com um valor *a priori* gaussiano $\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0) = N(\boldsymbol{\mu}_0, \Sigma_0)$, a filtragem com um modelo gaussiano linear produzirá uma distribuição de estados gaussiana para todo o tempo.

☞ Esse parece ser um resultado ótimo e elegante; porém, por que é tão importante? A razão é que, exceto para alguns casos especiais como esse, *a filtragem com redes contínuas ou híbridas*

(*discretas e contínuas*) *gera distribuições de estados cuja representação cresce sem limite ao longo do tempo*. Não é fácil provar essa declaração no caso geral, mas o Exercício 15.10 mostra o que acontece em um exemplo simples.

15.4.2 Um exemplo unidimensional simples

Dissemos que o operador PARAFRENTE para o filtro de Kalman mapeia um gaussiano em um novo gaussiano. Isso se traduz no cálculo de uma nova média e uma matriz de covariância a partir da média e da matriz de covariância anterior. A derivação da regra de atualização no caso geral (multivariado) exige bastante álgebra linear e, assim, vamos nos limitar por enquanto a um caso univariado muito simples; mais tarde daremos os resultados para o caso geral. Mesmo para o caso univariado, os cálculos são um tanto tediosos, mas achamos que vale a pena observá-los porque a utilidade do filtro de Kalman está amarrada de forma muito íntima às propriedades matemáticas de distribuições gaussianas.

O modelo temporal que consideraremos descreve um **percurso aleatório** de uma única variável de estado contínua X_t com uma observação com ruído Z_t. Um exemplo poderia ser o índice de "confiança do consumidor", que podemos modelar como um valor que sofre mudança aleatória com distribuição gaussiana a cada mês e que é medido por uma pesquisa aleatória entre os consumidores que também introduz ruído de amostragem gaussiano.

A distribuição anterior é considerada gaussiana com variância σ_0^2:

$$P(x_0) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)} ,$$

(Por simplicidade, usaremos o mesmo símbolo α para todas as constantes de normalização nesta seção.) O modelo de transição simplesmente adiciona uma perturbação gaussiana de variância constante σ_x^2 ao estado atual:

$$P(x_{t+1} \mid x_t) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_{t+1}-x_t)^2}{\sigma_x^2}\right)} .$$

O modelo de sensores então supõe um ruído gaussiano com a variância σ_z^2:

$$P(z_t \mid x_t) = \alpha\, e^{-\frac{1}{2}\left(\frac{(z_t-x_t)^2}{\sigma_z^2}\right)} .$$

Agora, dada a distribuição anterior $\mathbf{P}(X_0)$, podemos calcular a distribuição prevista de um passo usando a Equação 15.17:

$$P(x_1) = \int_{-\infty}^{\infty} P(x_1 \mid x_0)P(x_0)\,dx_0 = \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{(x_1-x_0)^2}{\sigma_x^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)}\,dx_0$$
$$= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{\sigma_0^2(x_1-x_0)^2+\sigma_x^2(x_0-\mu_0)^2}{\sigma_0^2\sigma_x^2}\right)}\,dx_0\,.$$

Essa integral parece bastante complicada. A chave para progredir é notar que o expoente é a soma de duas expressões que são *quadráticas* em x_0 e, consequentemente, ele próprio é quadrático em x_0. Um artifício simples conhecido como **completar o quadrado** permite a reescrita de qualquer expressão quadrática $ax_0^2 + bx_0 + c$ e $a(x_0 - \frac{-b}{2a})^2$ um termo residual $c - \frac{b^2}{4a}$ que é independente de x_0. O termo residual pode ser colocado fora da integral, o que nos dá:

$$P(x_1) = \alpha\, e^{-\frac{1}{2}\left(c-\frac{b^2}{4a}\right)} \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(a(x_0-\frac{-b}{2a})^2\right)}\,dx_0\,.$$

Agora, a integral é apenas a integral de um gaussiano sobre seu intervalo completo, que vale simplesmente 1. Desse modo, ficamos somente com o termo residual da expressão quadrática. Então, observamos que o termo residual é um quadrático em x_1; de fato, após a simplificação obtemos

$$P(x_1) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)}\,.$$

Ou seja, a distribuição prevista de um passo é um gaussiano com a mesma média μ_0 e uma variância igual à soma da variância original σ_0^2 e da variância de transição σ_x^2.

Para completar a etapa de atualização, precisamos fazer o condicionamento sobre a observação no primeiro período de tempo, ou seja, z_1. A partir da Equação 15.18, isso é dado por:

$$P(x_1 \mid z_1) = \alpha\, P(z_1 \mid x_1)P(x_1)$$
$$= \alpha\, e^{-\frac{1}{2}\left(\frac{(z_1-x_1)^2}{\sigma_z^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)}\,.$$

Mais uma vez, combinamos os expoentes e completamos o quadrado (Exercício 15.11), obtendo:

$$P(x_1 \mid z_1) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_1-\frac{(\sigma_0^2+\sigma_x^2)z_1+\sigma_z^2\mu_0}{\sigma_0^2+\sigma_x^2+\sigma_z^2})^2}{(\sigma_0^2+\sigma_x^2)\sigma_z^2/(\sigma_0^2+\sigma_x^2+\sigma_z^2)}\right)}\,. \qquad (15.19)$$

Desse modo, depois de um ciclo de atualização, temos uma nova distribuição gaussiana para a variável de estado.

A partir da fórmula gaussiana na Equação 15.19, vemos que a nova média e o novo desvio-padrão podem ser calculados a partir da média e do desvio-padrão antigos, assim:

$$\mu_{t+1} = \frac{(\sigma_t^2+\sigma_x^2)z_{t+1}+\sigma_z^2\mu_t}{\sigma_t^2+\sigma_x^2+\sigma_z^2} \quad \text{e} \quad \sigma_{t+1}^2 = \frac{(\sigma_t^2+\sigma_x^2)\sigma_z^2}{\sigma_t^2+\sigma_x^2+\sigma_z^2}\,. \qquad (15.20)$$

A Figura 15.10 mostra um ciclo de atualização para valores específicos dos modelos de transição e de sensores.

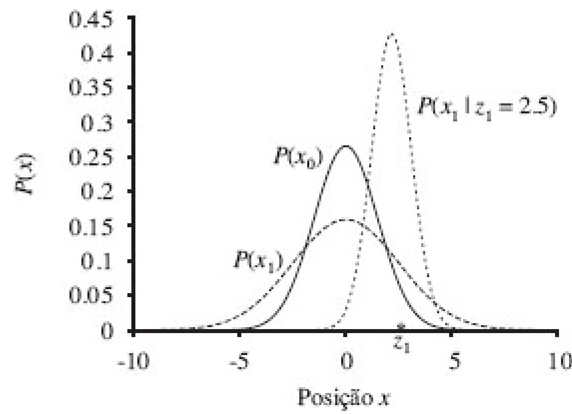


Figura 15.10 Fases no ciclo de atualização do filtro de Kalman para um percurso aleatório com distribuição anterior dada por m_0 = 0,0 e σ_0 = 1,0, ruído de transição dado por σ_x = 2,0, ruído de sensores dado por σ_z = 1,0 e uma primeira observação z_1 = 2,5 (marcada no eixo x). Note que a previsão $P(x_1)$ foi achatada, em relação a $P(x_0)$, pelo ruído de transição. Note também que a média da distribuição posterior $P(x_1|z_1)$ está ligeiramente à esquerda da observação z_1 porque a média é uma média ponderada entre a previsão e a observação.

A Equação 15.20 desempenha exatamente o mesmo papel da equação de filtragem geral (15.5) ou da equação de filtragem de MOM (15.12). Porém, devido à natureza especial das distribuições gaussianas, as equações têm algumas propriedades adicionais interessantes. Primeiro, podemos interpretar o cálculo para a nova média m_{t+1} simplesmente como uma *média ponderada* entre a nova observação z_{t+1} e a média antiga m_t. Se a observação é pouco confiável, σ_z^2 é grande e dedicamos maior atenção à média antiga; se a média antiga é pouco confiável (σ_t^2 é grande) ou se o processo é altamente imprevisível (σ_x^2 é grande), então dedicamos maior atenção à observação. Em segundo lugar, note que a atualização para a variância σ_{t+1}^2 é *independente da observação*. Podemos então calcular com antecedência qual será a sequência de valores de variância. Em terceiro lugar, a sequência de valores de variância converge rapidamente para um valor fixo que só depende de σ_x^2 e σ_z^2, simplificando assim de forma significativa os cálculos subsequentes (veja o o Exercício 15.12).

15.4.3 O caso geral

A derivação precedente ilustra a propriedade fundamental de distribuições gaussianas que permite o funcionamento da filtragem de Kalman: o fato de o expoente ser uma forma quadrática. Isso é verdadeiro não apenas para o caso univariado; a distribuição gaussiana multivariada completa tem a forma:

$$N(\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x}) = \alpha \, e^{-\frac{1}{2}\left((\mathbf{x}-\boldsymbol{\mu})^{\top}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)} .$$

A multiplicação dos termos no expoente torna claro que o expoente também é uma função

quadrática dos valores de x_i em **x**. Como no caso univariado, a atualização de filtragem preserva a natureza gaussiana da distribuição de estados.

Primeiro, vamos definir o modelo temporal geral usado com a filtragem de Kalman. Tanto o modelo de transição quanto o modelo de sensores permitem uma transformação *linear* com ruído gaussiano aditivo. Desse modo, temos:

$$\begin{aligned} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) &= N(\mathbf{F}\mathbf{x}_t, \mathbf{\Sigma}_x)(\mathbf{x}_{t+1}) \\ P(\mathbf{z}_t \mid \mathbf{x}_t) &= N(\mathbf{H}\mathbf{x}_t, \mathbf{\Sigma}_z)(\mathbf{z}_t) \ , \end{aligned} \qquad (15.21)$$

onde **F** e Σ_x são matrizes que descrevem o modelo de transição linear e a covariância de ruído de transição, e onde **H** e Σ_z são as matrizes correspondentes para o modelo de sensores. Agora, as equações de atualização para a média e a covariância, em sua forma total e extremamente complicada, são:

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \mathbf{F}\boldsymbol{\mu}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t) \\ \mathbf{\Sigma}_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x) \ , \end{aligned} \qquad (15.22)$$

onde $\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top + \mathbf{\Sigma}_z)^{-1}$ é chamada **matriz de ganho de Kalman**. Acredite ou não, essas equações fazem algum sentido intuitivo. Por exemplo, considere a atualização para a estimativa de estado médio $\boldsymbol{\mu}$. O termo $\boldsymbol{F\mu}_t$ é o estado *previsto* em $t + 1$ e, assim, $\boldsymbol{HF\mu}_t$ é a observação *prevista*. Portanto, o termo $\mathbf{z}_{t+1} - \mathbf{HF}\boldsymbol{\mu}t$ representa o erro na observação prevista. Esse termo é multiplicado por $\mathbf{K}_{t+1}$ para corrigir o estado previsto; consequentemente, $\mathbf{K}_{t+1}$ é uma medida do *quanto devemos levar a sério a nova observação* em relação à previsão. Como na Equação 15.18, também temos a propriedade de que a atualização da variância é independente das observações. A sequência de valores para Σ_t e $\mathbf{K}_t$ pode então ser calculada off-line, e os cálculos reais exigidos durante o acompanhamento on-line serão bastante modestos.

Para ilustrar essas equações em funcionamento, elas foram aplicadas ao problema de acompanhar um objeto em movimento no plano *X*−*Y*. As variáveis de estados são $\mathbf{X} = (X, Y, \dot{X}, \dot{Y})^\top$ e, assim, **F**, Σ_x, **H** e Σ_z são matrizes 4 × 4. A Figura 15.11(a) mostra a trajetória verdadeira, uma série de observações com ruído e a trajetória estimada pela filtragem de Kalman, juntamente com as covariâncias indicadas pelos contornos do único desvio-padrão. O processo de filtragem faz um bom trabalho de acompanhamento do movimento real e, como esperado, a variância alcança rapidamente um ponto fixo.

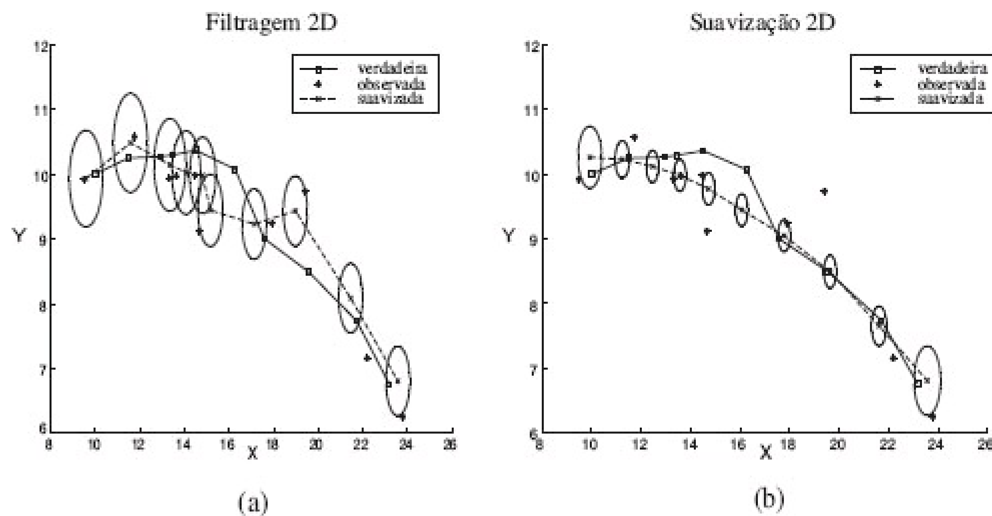


Figura 15.11 (a) Resultados de filtragem de Kalman para um objeto em movimento no plano *X*−*Y*, mostrando a trajetória verdadeira (da esquerda para a direita), uma série de observações com ruído e a trajetória estimada por filtragem de Kalman. A variância na estimativa de posição é indicada pelas elipses. (b) Resultados de suavização de Kalman para a mesma sequência de observação.

Também podemos derivar equações para *suavização*, bem como para filtragem com modelos gaussianos lineares. Os resultados de suavização são mostrados na Figura 15.11(b). Note que a variância na estimativa de posição é nitidamente reduzida, exceto nas extremidades da trajetória (por quê?) e que a trajetória estimada é muito mais suave.

15.4.4 A aplicabilidade da filtragem de Kalman

A filtragem de Kalman e suas elaborações são usadas em uma vasta série de aplicações. A aplicação "clássica" de filtros de Kalman ocorre em acompanhamento por radar de aeronaves e mísseis. Aplicações inter-relacionadas incluem acompanhamento acústico de submarinos e veículos terrestres e, ainda, acompanhamento visual de veículos e pessoas. Em um sentido ligeiramente mais esotérico, os filtros de Kalman são utilizados para reconstruir trajetórias de partículas a partir de fotografias de câmaras de bolha e correntes oceânicas a partir de medições de superfícies feitas por satélites. A variedade de aplicações é muito maior que o simples acompanhamento de movimentos: qualquer sistema caracterizado por variáveis de estados contínuos e medições com ruído fará uso desse recurso. Tais sistemas incluem fábricas de polpa, indústrias químicas, reatores nucleares, ecossistemas vegetais e economias nacionais.

O fato de ser possível aplicar a filtragem de Kalman a um sistema não significa que os resultados serão válidos ou úteis. As hipóteses assumidas — uma transição gaussiana linear e modelos de sensores — são muito fortes. O **filtro de Kalman estendido** (**FKE**) tenta superar não linearidades no sistema que está sendo modelado. Um sistema é não linear se o modelo de transição não pode ser descrito como uma multiplicação de matrizes do vetor de estados, como na Equação 15.21. O FKE funciona modelando o sistema como *localmente* linear em $\mathbf{x}_t$ na região de $\mathbf{x}_t = \boldsymbol{\mu}_t$, a média da distribuição de estados atual. Isso funciona bem para sistemas suaves e bem comportados, e permite ao controlador manter e atualizar uma distribuição de estados gaussiana que representa uma aproximação razoável para a distribuição posterior verdadeira. Há um exemplo detalhado no

Capítulo 25.

O que significa um sistema ser "não suave" ou "mal comportado"? Tecnicamente, isso quer dizer que existe uma não linearidade significativa na resposta do sistema dentro da região que está "próxima" (de acordo com a covariância Σ_t) à média corrente $\boldsymbol{\mu}_t$. Para compreender essa ideia em termos não técnicos, considere o exemplo de tentar localizar um pássaro à medida que ele voa pela floresta. O pássaro parece estar se dirigindo em alta velocidade para um tronco de árvore. O filtro de Kalman, seja ele regular ou estendido, só pode fazer uma previsão gaussiana da posição do pássaro, e a média dessa previsão gaussiana estará centrada no tronco, como mostra a Figura 15.12(a). Por outro lado, um modelo razoável do pássaro iria prever uma ação evasiva para um lado ou outro, como mostra a Figura 15.12(b). Tal modelo é altamente não linear porque a decisão do pássaro varia nitidamente, dependendo de sua posição exata em relação ao tronco.

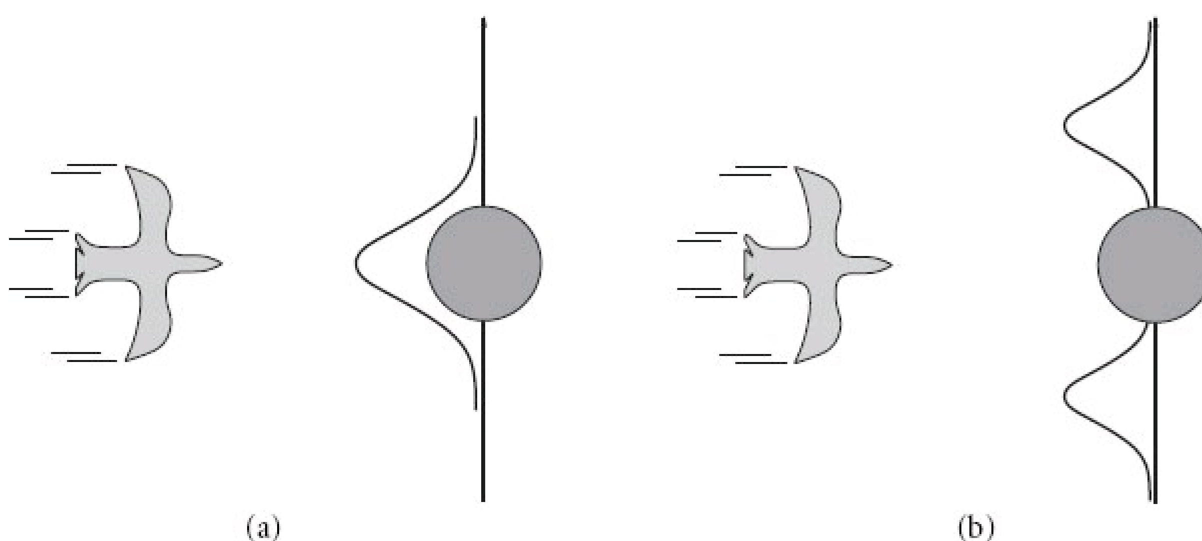


Figura 15.12 Um pássaro que voa em direção a uma árvore (vistas superiores). (a) Um filtro de Kalman vai prever a posição do pássaro usando um único gaussiano centrado no obstáculo. (b) Um modelo mais realista permite uma ação evasiva do pássaro, prevendo que ele voará para um lado ou para o outro.

Para tratar exemplos como esses, é claro que precisamos de uma linguagem mais expressiva para representar o comportamento do sistema que está sendo modelado. Dentro da comunidade de teoria de controle, para a qual problemas como manobras evasivas de aeronaves encontram os mesmos tipos de dificuldades, a solução-padrão é o **filtro de Kalman de comutação**. Nessa abordagem, vários filtros de Kalman funcionam em paralelo, cada um usando um modelo diferente do sistema — por exemplo, um para voo em linha reta, um para curvas bruscas à esquerda e um para curvas bruscas à direita. É usada uma soma ponderada de previsões em que o peso depende do quanto cada filtro se adapta bem aos dados atuais. Veremos na próxima seção que esse é simplesmente um caso especial do modelo geral de rede bayesiana dinâmica, obtido pela adição de uma variável de estado discreta de "manobra" à rede mostrada na Figura 15.9. Os filtros de Kalman de comutação serão descritos com mais detalhes no Exercício 15.10.

15.5 REDES BAYESIANAS DINÂMICAS

Uma **rede bayesiana dinâmica**, ou **DBN**, é uma rede bayesiana que representa um modelo de probabilidade temporal do tipo descrito na Seção 15.1. Já vimos exemplos de DBNs: a rede de guarda-chuva da Figura 15.2 e a rede de filtro de Kalman da Figura 15.9. Em geral, cada fatia de uma DBN pode ter qualquer número de variáveis de estados $\mathbf{X}_t$ e variáveis de evidência $\mathbf{E}_t$. Por simplicidade, vamos supor que as variáveis e seus vínculos são reproduzidos exatamente de uma fatia para outra e que a DBN representa um processo de Markov de primeira ordem, de forma que cada variável possa ter pais somente em sua própria fatia ou na fatia imediatamente precedente.

Deve ficar claro que todo modelo oculto de Markov pode ser representado como uma DBN com uma única variável de estado e uma única variável de evidência. Também ocorre que toda DBN de variáveis discretas pode ser representada como um MOM; conforme explicamos na Seção 15.3, podemos combinar todas as variáveis de estados na DBN em uma única variável de estado cujos valores são todas as tuplas de valores possíveis das variáveis de estados individuais. Agora, se todo MOM for uma DBN e toda DBN puder ser convertida em um MOM, qual será a diferença? A diferença é que, *decompondo-se o estado de um sistema complexo em suas variáveis constituintes*, *a DBN será capaz de tirar proveito da* escassez *no modelo de probabilidade temporal*. Por exemplo, suponha que uma DBN tenha 20 variáveis de estados booleanas, cada uma das quais tem três pais na fatia precedente. Então, o modelo de transição de DBN tem $20 \times 2^3 = 160$ probabilidades, enquanto o MOM correspondente tem 2^{20} estados e, portanto, 2^{40}, ou, aproximadamente, um trilhão de probabilidades na matriz de transição. Isso é ruim por pelo menos três razões: primeiro, o próprio MOM exige muito mais espaço; em segundo lugar, a enorme matriz de transição torna a inferência de MOM muito mais dispendiosa; em terceiro lugar, o problema de aprender um número tão enorme de parâmetros torna o modelo MOM puro inadequado para problemas grandes. O relacionamento entre DBNs e MOMs é aproximadamente análogo ao relacionamento entre redes bayesianas comuns e distribuições conjuntas totalmente tabuladas.

Já explicamos que todo modelo de filtro de Kalman pode ser representado em uma DBN com variáveis contínuas e distribuições condicionais gaussianas lineares (Figura 15.9). A partir da discussão do final da seção precedente, deve ficar claro que *nem toda* DBN pode ser representada por um modelo de filtro de Kalman. Em um filtro de Kalman, a distribuição de estados atual é sempre uma única distribuição gaussiana multivariada, isto é, um único "impacto" em uma posição específica. Por outro lado, as DBNs podem modelar distribuições arbitrárias. Em muitas aplicações reais, essa flexibilidade é essencial. Por exemplo, considere a posição atual de minhas chaves. Elas poderiam estar em meu bolso, na mesa de cabeceira, sobre a mesa da cozinha, penduradas na fechadura da porta da frente ou trancadas no carro. Uma colina gaussiana única que incluísse todos esses lugares teria de alocar uma probabilidade significativa de que as chaves estivessem suspensas no ar no corredor da frente. Aspectos do mundo real, como agentes intencionais, obstáculos e bolsos, introduzem "não linearidades" que exigem combinações de variáveis discretas e contínuas, a fim de obter modelos razoáveis.

15.5.1 Construção de DBNs

Para construir uma DBN, devemos especificar três tipos de informações: a distribuição anterior

sobre as variáveis de estados, $\mathbf{P}(\mathbf{X}_0)$; o modelo de transição $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t)$; e o modelo de sensores $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$. Para especificar os modelos de transição e de sensores, também devemos especificar a topologia das conexões entre fatias sucessivas e entre as variáveis de estados e de evidência. Como os modelos de transição e de sensores são supostamente estacionários — os mesmos para todo t —, é mais conveniente simplesmente especificá-los para a primeira fatia. Por exemplo, a especificação de DBN completa para o mundo do guarda-chuva é dada pela rede de três nós da Figura 15.13(a). A partir dessa especificação, é possível construir a DBN completa com um número infinito de fatias de tempo, conforme seja necessário, copiando a primeira fatia.

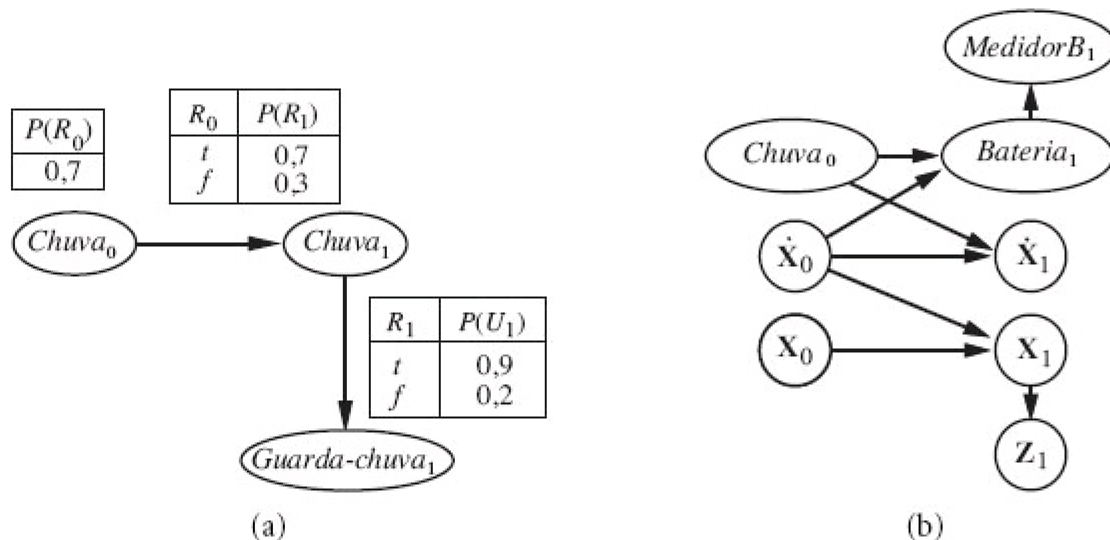


Figura 15.13 (a) Especificação do modelo de transição anterior e do modelo de sensores para a DBN de guarda-chuva. Todas as fatias subsequentes são consideradas cópias das fatia 1. (b) DBN simples para o movimento de um robô no plano X−Y.

Agora, vamos considerar um exemplo mais interessante: o monitoramento de um robô alimentado por bateria que se move no plano X−Y, como vimos no final da Seção 15.1. Primeiro, precisamos de variáveis de estados, que incluirão tanto $\mathbf{X}_t = (X_t, Y_t)$ para representar a posição quanto $\dot{X}_t = (\dot{X}_t, \dot{Y}_t)$ para representar a velocidade.

Presumiremos algum método para medir a posição — talvez uma câmera fixa ou um GPS (*Global Positioning System*) a bordo — produzindo medições de $\mathbf{Z}_t$. A posição no período de tempo seguinte dependerá da posição atual e da velocidade, como no modelo de filtro de Kalman padrão. A velocidade no período seguinte dependerá da velocidade atual e do estado da bateria. Acrescentamos $Bateria_t$ para representar o nível de carga real da bateria, que tem como pais o nível da bateria anterior e a velocidade, e também adicionamos $MedidorB_t$, que mede o nível de carga da bateria. Isso nos dá o modelo básico mostrado na Figura 15.13(b).

Vale a pena examinarmos com maior profundidade a natureza do modelo de sensores correspondente a $MedidorB_t$. Vamos supor, por simplicidade, que tanto $Bateria_t$ quanto $MedidorB_t$ possam assumir valores discretos de 0 até 5, de modo muito semelhante ao medidor do nível da bateria em um laptop típico. Se o medidor for sempre preciso, a TPC (tabela de probabilidade condicional) $\mathbf{P}(MedidorB_t \mid Bateria_t)$ deve ter probabilidades 1,0 "ao longo da diagonal" e probabilidades 0,0 nos outros lugares. Na realidade, o ruído sempre interfere nas medições. No caso

de medições contínuas, poderia ser usada em vez disso uma distribuição gaussiana com pequena variância.[5] No caso de nossas variáveis discretas, podemos fazer a aproximação de uma gaussiana utilizando uma distribuição na qual a probabilidade de erro caia de maneira apropriada, de modo que a probabilidade de um erro grande seja muito pequena. Utilizaremos a expressão **modelo de erro gaussiano** para cobrir tanto a versão contínua quanto a versão discreta.

Qualquer pessoa com experiência prática em robótica, controle de processos computadorizados ou outras formas de detecção automática admitirá prontamente o fato de que pequenas quantidades de ruído de medição são muitas vezes o menor dos problemas. Os sensores reais *falham*. Quando um sensor falha, ele não envia necessariamente um sinal afirmando: "Oh, a propósito, os dados que estou prestes a lhe enviar não têm o menor sentido." Em vez disso, ele simplesmente envia os dados sem sentido. A espécie mais simples de falha é chamada **falha transiente**, na qual o sensor decide ocasionalmente enviar alguns dados sem sentido. Por exemplo, o sensor de nível da bateria pode ter o hábito de enviar um zero quando alguém se choca com o robô, mesmo que a bateria esteja completamente carregada.

Vamos ver o que acontece quando ocorre uma falha transiente com um modelo de erro gaussiano que não admite tais falhas. Por exemplo, suponha que o robô esteja calmamente sentado e observe 20 leituras consecutivas da bateria indicando o nível de carga 5. Em seguida, o medidor da bateria tem um ataque temporário e a leitura seguinte é $MedidorB_{21} = 0$. Qual será o modelo de erro gaussiano simples que nos levará a acreditar em $Bateria_{21}$? De acordo com a regra de Bayes, a resposta depende tanto do modelo de sensores $\mathbf{P}(MedidorB_{21} = 0 \mid Bateria_{21})$ quanto da previsão $\mathbf{P}(Bateria_{21} \mid MedidorB_{1:20})$. Se a probabilidade de um grande erro de sensor for significativamente menos provável que a probabilidade de uma transição para $Bateria_{21} = 0$, mesmo que esta última seja muito improvável, então a distribuição posterior atribuirá alta probabilidade ao fato de a bateria estar sem carga. Uma segunda leitura igual a zero em $t = 22$ tornará essa conclusão quase certa. Se a falha transiente desaparecer em seguida e a leitura retornar a 5 a partir de $t = 23$ em diante, a estimativa do nível da bateria voltará rapidamente a 5, como se fosse por mágica. Esse curso de eventos é ilustrado na curva superior da Figura 15.14(a), que mostra o valor esperado de $Bateria_t$ ao longo do tempo, usando um modelo de erro gaussiano discreto.

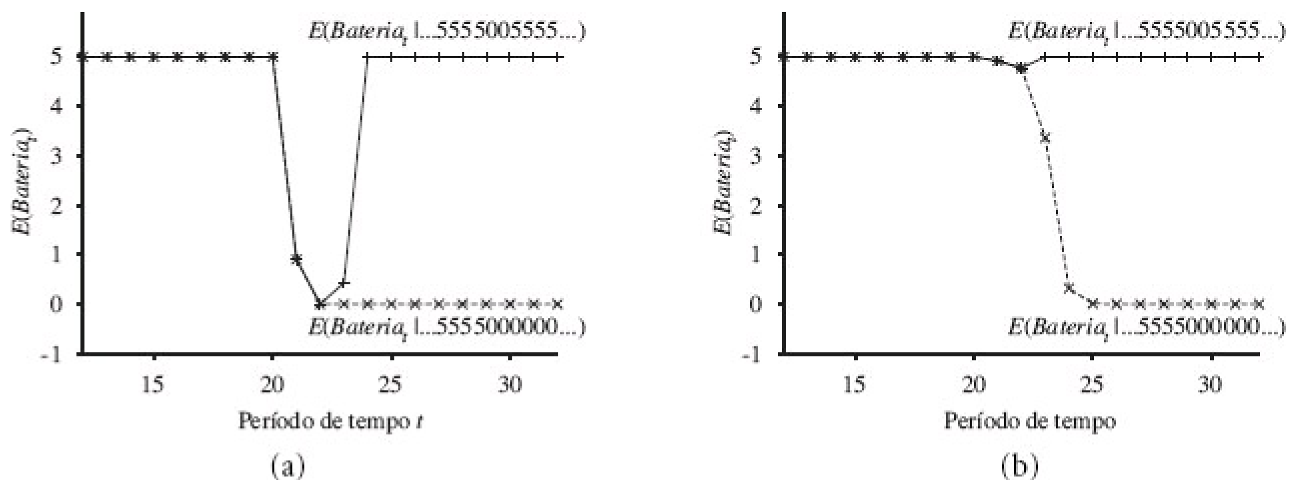


Figura 15.14 (a) Curva superior: trajetória do valor esperado de $Bateria_t$ para uma sequência de observações que consiste apenas em valores 5, exceto pelos valores 0 em $t = 21$ e $t = 22$, usando um modelo de erro gaussiano simples. Curva inferior: trajetória quando a observação permanece igual a

0 a partir de $t = 21$. (b) A mesma experiência com o modelo de falha transiente. Note que a falha transiente é bem tratada, mas a falha persistente resulta em pessimismo excessivo sobre a carga da bateria.

Apesar da recuperação, existe um tempo ($t = 22$) em que o robô se convence de que sua bateria está sem carga; então, presume-se que ele deve enviar um sinal de socorro e se desligar. Infelizmente, seu modelo de sensores supersimplificado fez com que ele se perdesse. Como isso pode ser corrigido? Considere um exemplo familiar extraído da atividade humana diária de dirigir: em curvas bruscas ou em colinas íngremes, o "tanque de combustível se esvazia" e às vezes faz a luz de advertência acender. Em vez de procurar pelo telefone de emergência, lembramos que o medidor de nível de combustível simplesmente comete um erro muito grande quando o combustível está jogando de um lado para outro dentro do tanque. Moral da história é: *para o sistema manipular falhas de sensores de maneira apropriada, o modelo de sensores deve incluir a possibilidade de falha*.

A espécie mais simples de modelo de falha para um sensor dá certa margem de probabilidade de que o sensor retornará algum valor completamente incorreto, independentemente do estado verdadeiro do mundo. Por exemplo, se o medidor de bateria falhar retornando 0, poderemos dizer que

$$P(MedidorB_t = 0 \mid Bateria_t = 5) = 0{,}03,$$

que presumivelmente é muito maior que a probabilidade atribuída pelo modelo de erro gaussiano simples. Vamos chamá-lo **modelo de falha transiente**. De que maneira ele nos ajuda quando estamos diante de uma leitura igual a 0? Considerando-se que a probabilidade *prevista* de uma bateria sem carga, de acordo com as leituras feitas até o momento, é muito menor que 0,03, a melhor explicação da observação $MedidorB_{21} = 0$ é que o sensor falhou temporariamente. Por intuição, podemos imaginar que a crença sobre o nível da bateria tem certa quantidade de "inércia" que ajuda a superar oscilações temporárias na leitura do medidor. A curva superior da Figura 15.14(b) mostra que o modelo de falha transiente pode manipular falhas transientes sem mudança catastrófica nas crenças.

Essa é a situação no caso de oscilações temporárias. E, se houver falha persistente de um sensor? Infelizmente, falhas desse tipo são bastante comuns. Se o sensor retornar 20 leituras iguais a 5 seguidas por 20 leituras iguais a 0, então o modelo de falha de sensor transiente descrito no parágrafo anterior terá como resultado o fato de o robô ser levado gradualmente a acreditar que sua bateria está sem carga quando de fato talvez o medidor tenha falhado. A curva inferior da Figura 15.14(b) mostra a "trajetória" de crença para esse caso. Depois de $t = 25$ — cinco leituras iguais a 0 —, o robô se convence de que sua bateria está sem carga. É óbvio que preferiríamos que o robô acreditasse que o medidor de sua bateria está quebrado, se esse de fato fosse o evento mais provável.

Não surpreende que, para tratar uma falha persistente, tenhamos necessidade de um **modelo de falha persistente** que descreva como o sensor se comporta sob condições normais e após uma falha. Para isso, precisamos ampliar o estado oculto do sistema com uma variável adicional, digamos *MBQuebrado*, que descreva o *status* do medidor da bateria. A persistência da falha deve ser

modelada por um arco vinculando $MBQuebrado_0$ a $MBQuebrado_1$. Esse **arco de persistência** tem uma TPC que fornece pequena probabilidade de falha em qualquer período de tempo dado, digamos 0,001, mas especifica que o sensor permanece quebrado depois de se quebrar. Quando o sensor está em perfeitas condições, o modelo de sensores para *MedidorB* é idêntico ao modelo de falha transiente; quando o sensor está quebrado, ele informa que *MedidorB* é sempre 0, independentemente do estado real da bateria.

O modelo de falha persistente para o sensor de bateria é mostrado na Figura 15.15(a). Seu desempenho sobre as duas sequências de dados (oscilação temporária e falha persistente) é mostrado na Figura 15.15(b). Devemos notar vários detalhes sobre essas curvas. Primeiro, no caso da oscilação temporária, a probabilidade de que o sensor esteja quebrado se eleva de forma significativa após a segunda leitura 0, mas volta a cair imediatamente a zero depois de se observar um valor 5. Em segundo lugar, no caso de falha persistente, a probabilidade de que o sensor esteja quebrado se eleva com rapidez até um valor próximo de 1 e permanece com esse valor. Por fim, uma vez confirmado que o sensor está quebrado, o robô só pode supor que sua bateria se descarrega à velocidade "normal", como mostra o nível gradualmente descendente de $E(Bateria_t \mid \ldots)$.

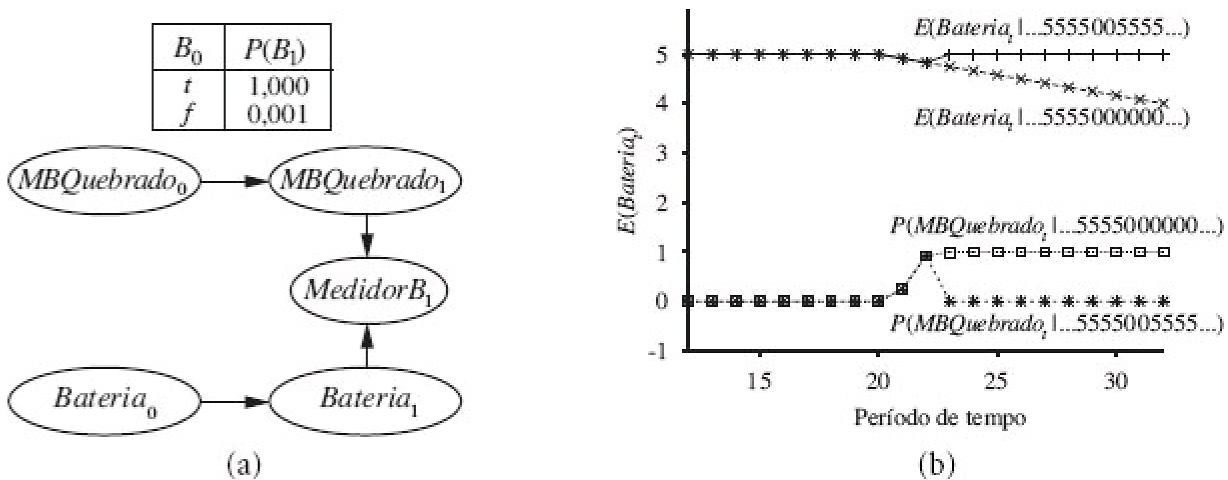


Figura 15.15 (a) Um fragmento de DBN mostrando a variável de *status* de sensor exigida para a modelagem de falha persistente do sensor da bateria. (b) Curvas superiores: trajetórias do valor esperado de $Bateria_t$ para as sequências de observações de "falha transiente" e "falha permanente". Curvas inferiores: trajetórias de probabilidade para *MBQuebrado*, dadas as duas sequências de observações.

Até agora, apenas arranhamos a superfície do problema de representação de processos complexos. A variedade de modelos de transição é enorme, englobando tópicos tão discrepantes quanto a modelagem do sistema endócrino humano e a modelagem de vários veículos trafegando em uma autoestrada. A modelagem de sensores também é por si só um vasto subcampo, mas, mesmo fenômenos sutis, como flutuação de sensores, descalibração repentina e os efeitos de condições exógenas (como as condições do clima) sobre leituras de sensores, podem ser manipuladas por representação explícita dentro de redes bayesianas dinâmicas.

15.5.2 Inferência exata em DBNs

Tendo esboçado algumas ideias para representar processos complexos como DBNs, vamos passar agora à questão da inferência. De certo modo, essa pergunta já foi respondida: as redes bayesianas dinâmicas *são* redes bayesianas, e já temos algoritmos para inferência em redes bayesianas. Dada uma sequência de observações, é possível construir a representação de rede bayesiana total de uma DBN replicando fatias até a rede ser grande o bastante para acomodar as observações, como na Figura 15.16. Essa técnica, mencionada no Capítulo 14 no contexto dos modelos de probabilidade relacional, é chamada **desenrolamento**. (Tecnicamente, a DBN é equivalente à rede semi-infinita obtida por desenrolamento eterno. Fatias adicionadas depois da última observação não têm nenhum efeito sobre inferências dentro do período de observação e podem ser omitidas.) Uma vez que a DBN é desenvolvida, pode-se empregar qualquer dos algoritmos de inferência — eliminação de variáveis, métodos de árvores de junção, e assim por diante — descritos no Capítulo 14.

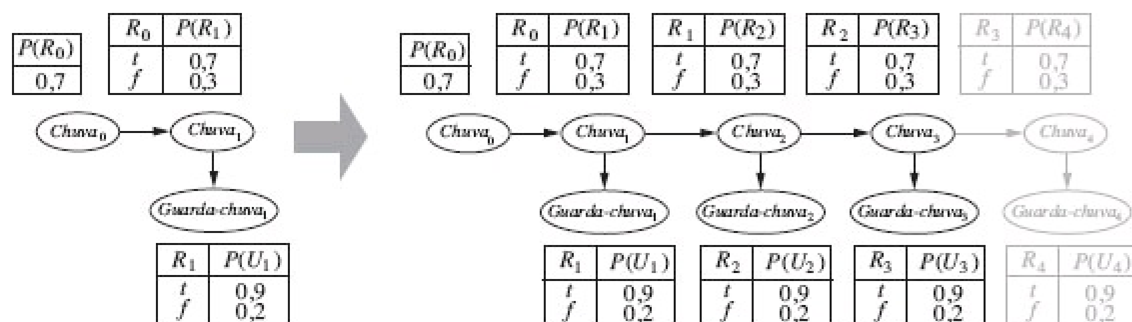


Figura 15.16 Desenrolamento de uma rede bayesiana dinâmica: as fatias são replicadas para acomodar a sequência de observações *gurda-chuva*$_{1:3}$. Fatias adicionais não têm nenhum efeito sobre inferências dentro do período de observação.

Infelizmente, uma aplicação ingênua de desenrolamento não seria particularmente eficiente. Se quiséssemos executar a filtragem ou a suavização com uma longa sequência de observações $\mathbf{e}_{1:t}$, a rede desenrolada exigiria o espaço $O(t)$ e, portanto, cresceria sem limite à medida que fossem adicionadas mais observações. Além disso, se simplesmente executarmos de novo o algoritmo de inferência a cada vez que for adicionada uma observação, o tempo de inferência por atualização também aumentará com $O(t)$.

Voltando à Seção 15.2.1, vemos que é possível obter atualização com tempo e espaço constantes por filtragem se a computação puder ser realizada de forma recursiva. Em essência, a atualização de filtragem na Equação 15.5 funciona *efetuando-se o somatório* das variáveis de estados do período de tempo anterior, a fim de se obter a distribuição correspondente ao novo período de tempo. Efetuar o somatório das variáveis é exatamente o que faz o algoritmo de **eliminação de variáveis** (Figura 14.11), e ocorre que a execução da eliminação de variáveis com as variáveis em ordem temporal imita de maneira exata a operação da atualização de filtragem recursiva da Equação 15.5. O algoritmo modificado mantém, no máximo, duas fatias na memória em qualquer instante: começando com a fatia 0, adicionamos a fatia 1, depois somamos a fatia 0, em seguida adicionamos a fatia 2, somamos a fatia 1, e assim por diante. Desse modo, podemos conseguir atualização com espaço e tempo constantes por filtragem (o mesmo desempenho pode ser alcançado fazendo-se modificações apropriadas no agrupamento do algoritmo). O Exercício 15.17 pede para verificar esse fato no caso da rede de guarda-chuva.

Agora esgotamos as boas notícias; vamos então examinar as más notícias. Ocorre que a

"constante" para complexidade de tempo e espaço por atualização é, em quase todos os casos, exponencial em relação ao número de variáveis de estados. Acontece que, à medida que a eliminação de variáveis prossegue, os fatores crescem até incluir todas as variáveis de estados (ou, mais precisamente, todas as variáveis de estados que têm pais na fatia de tempo anterior). O tamanho máximo do fator é $O(d^{n+k})$ e o custo da atualização por etapa é $O(nd^{n+k})$, onde d é o tamanho do domínio das variáveis e k é o número máximo de pais de qualquer variável de estado.

É claro que esse é um valor muito menor que o custo de atualização de MOM, que é $O(d^{2n})$, mas ainda é inviável para grande número de variáveis. Esse terrível fato é algo um tanto difícil de aceitar. Ele significa que, *embora possamos usar DBNs para* representar *processos temporais muito complexos com muitas variáveis esparsamente conectadas, não podemos* raciocinar *de modo eficiente e exato sobre esses processos*. O próprio modelo de DBN, que representa a distribuição conjunta anterior sobre todas as variáveis, pode ser fatorado em suas TPCs constituintes, mas a distribuição conjunta posterior condicionada sobre uma sequência de observações — ou seja, a mensagem para a frente — em geral *não* é fatorável. Até agora, ninguém encontrou um modo de contornar esse problema, a despeito do fato de que muitas áreas importantes de ciência e engenharia se beneficiariam enormemente de sua solução. Desse modo, devemos recorrer a métodos aproximados.

15.5.3 Inferência aproximada em DBNs

A Seção 14.5 descreveu dois algoritmos de aproximação: ponderação de probabilidades (Figura 14.15) e cadeia de Markov Monte Carlo (CMMC, Figura 14.16). Dos dois, o primeiro se adapta com maior facilidade ao contexto de DBN (um algoritmo de filtragem CMMC será descrito brevemente nas notas ao final do capítulo). No entanto, veremos que são necessários vários aperfeiçoamentos sobre o algoritmo-padrão de ponderação de probabilidades antes de emergir um método prático.

Lembre-se de que a ponderação de probabilidades funciona por amostragem dos nós de não evidência da rede em ordem topológica, ponderando cada amostra pela probabilidade que ela concede às variáveis de evidência observadas. Como ocorre no caso de algoritmos exatos, poderíamos aplicar a ponderação de probabilidades diretamente a uma DBN não desenrolada, mas isso acarretaria os mesmos problemas em termos de aumento de requisitos de tempo e espaço por atualização, à medida que a sequência de observações crescesse. O problema é que o algoritmo-padrão executa cada amostra por sua vez, percorrendo toda a rede. Em vez disso, podemos simplesmente executar todas as N amostras juntas pela DBN, uma fatia de cada vez. O algoritmo modificado se ajusta ao padrão geral de algoritmos de filtragem, com o conjunto de N amostras como a mensagem para a frente. Então, a primeira inovação-chave é *usar as próprias amostras como uma representação aproximada da distribuição de estados atual*. Isso atende ao requisito de um tempo "constante" por atualização, embora a constante dependa do número de amostras necessárias para manter uma aproximação razoável em relação à distribuição posterior verdadeira. Não há nenhuma necessidade de desenrolar a DBN porque precisamos ter na memória apenas a fatia atual e a próxima fatia.

Em nossa discussão da ponderação de probabilidades no Capítulo 14, destacamos que a exatidão

do algoritmo sofre se as variáveis de evidência estão “abaixo” das variáveis que estão sendo amostradas porque, nesse caso, as amostras são geradas sem qualquer influência da evidência. Examinando a estrutura típica de uma DBN — digamos, a DBN do guarda-chuva da Figura 15.16 — vemos que, na verdade, as primeiras variáveis de estados serão amostradas sem o benefício da evidência posterior. De fato, observando com maior cuidado, vemos que *nenhuma* das variáveis de estados tem *quaisquer* variáveis de evidência entre seus ancestrais! Consequentemente, embora o peso de cada amostra dependa da evidência, o conjunto real de amostras geradas será *completamente independente* da evidência. Por exemplo, ainda que o chefe trouxesse o guarda-chuva todo dia, o processo de amostragem poderia refletir infinitos dias de sol. Na prática, isso significa que a fração de amostras que permanecem razoavelmente próximas à série real de eventos (e, portanto, têm pesos não elegíveis) cai exponencialmente com t, a duração da sequência de observações; em outras palavras, para manter dado nível de exatidão, precisamos aumentar exponencialmente o número de amostras com t. Considerando que um algoritmo de filtragem que funcione em tempo real só pode usar um número fixo de amostras, o efeito prático é que o erro explode depois de um número muito pequeno de passos de atualização.

Sem dúvida, precisamos de uma solução melhor. A segunda inovação-chave é *concentrar o conjunto de amostras nas regiões de alta probabilidade do espaço de estados*. Isso pode ser feito descartando-se amostras que têm peso muito baixo, de acordo com as observações, enquanto se replicam as que têm peso elevado. Desse modo, a população de amostras permanecerá razoavelmente próxima da realidade. Se pensarmos nas amostras como um recurso para modelar a distribuição posterior, fará sentido usar mais amostras em regiões do espaço de estados em que a distribuição posterior for mais alta.

Uma família de algoritmos denominados algoritmos de **filtragem de partículas** foi projetada para fazer exatamente isso. A filtragem de partículas funciona assim: primeiro, uma população de N amostras de estado inicial é criada por amostragem da distribuição anterior no tempo 0, $\mathbf{P}(\mathbf{X}_0)$. Em seguida, o ciclo de atualização é repetido para cada período de tempo:

1. Cada amostra é propagada para a frente por amostragem do próximo valor de estado $\mathbf{x}_{t+1}$, dado o valor atual $\mathbf{x}_t$ para a amostra, e usando-se o modelo de transição $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)$.
2. Cada amostra é ponderada pela probabilidade que atribui à nova evidência, $P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1})$.
3. A população é *reamostrada* para gerar uma nova população de N amostras. Cada nova amostra é selecionada a partir da população atual; a probabilidade de uma amostra específica ser selecionada é proporcional ao seu peso. As novas amostras são não ponderadas.

O algoritmo é mostrado em detalhes na Figura 15.17, e sua operação para a DBN do guarda-chuva é ilustrada na Figura 15.18.

função FILTRAGEM-DE-PARTÍCULAS(**e**, N, *dbn*) **retorna** um conjunto de amostras para o próximo período de tempo
entradas: **e**, a nova evidência de entrada
N, o número de amostras a serem mantidas
dbn, uma DBN com modelo de transição anterior $\mathbf{P}(\mathbf{X}_0)$, modelo de transição $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0)$ e modelo de sensores $\mathbf{P}(\mathbf{E}_1|\mathbf{X}_1)$
persistente: S, um vetor de amostras de tamanho N, inicialmente gerado a partir de $\mathbf{P}(\mathbf{X}_0)$
variáveis locais: W, um vetor de pesos de tamanho N

para i = 1 **até** N **faça**
$S[i] \leftarrow$ amostra de $\mathbf{P}(\mathbf{X}_1 \mid X_0 = S[i])$ /*etapa 1 */
$W[i] \leftarrow \mathbf{P}(\mathbf{e} \mid \mathbf{X}_1 = S[i])$ /*etapa 2 */
$S \leftarrow$ AMOSTRAGEM-PONDERADA-COM-REPOSIÇÃO(N, S, W) /*etapa 3 */
retornar S

função FILTRAGEM-DE-PARTÍCULAS(**e**, N, *dbn*) **retorna** um conjunto de amostras para o próximo período de tempo
entradas: **e**, a nova evidência de entrada
N, o número de amostras a serem mantidas
dbn, uma DBN com modelo de transição anterior $\mathbf{P}(\mathbf{X}_0)$, modelo de transição $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0)$ e modelo de sensores $\mathbf{P}(\mathbf{E}_1|\mathbf{X}_1)$
persistente: S, um vetor de amostras de tamanho N, inicialmente gerado a partir de $\mathbf{P}(\mathbf{X}_0)$
variáveis locais: W, um vetor de pesos de tamanho N

para i = 1 **até** N **faça**
$S[i] \leftarrow$ amostra de $\mathbf{P}(\mathbf{X}_1 \mid X_0 = S[i])$ /*etapa 1 */
$W[i] \leftarrow \mathbf{P}(\mathbf{e} \mid \mathbf{X}_1 = S[i])$ /*etapa 2 */
$S \leftarrow$ AMOSTRAGEM-PONDERADA-COM-REPOSIÇÃO(N, S, W) /*etapa 3 */
retornar S

Figura 15.17 Algoritmo de filtragem de partículas implementado como uma operação de atualização recursiva com estado (o conjunto de amostras). Cada uma das etapas de amostragem envolve a amostragem das variáveis de fatias relevantes em ordem topológica, de modo muito semelhante à AMOSTRAGEM-*A-PRIORI*. A operação de AMOSTRAGEM-PONDERADA-COM-REPOSIÇÃO pode ser implementada para funcionar no tempo esperado $O(N)$. O número de etapas refere-se à descrição no texto.

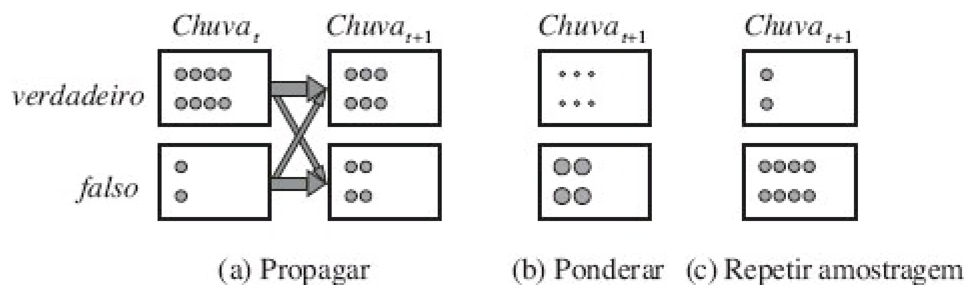


Figura 15.18 Ciclo de atualização da filtragem de partículas para a DBN de guarda-chuva com N = 10, com as populações de amostras de cada estado. (a) No tempo t, oito amostras indicam *chuva* e duas indicam ¬*chuva*. Cada uma é propagada para a frente pela amostragem do estado seguinte via

modelo de transição. No tempo $t + 1$, 6 amostras indicam *chuva* e quatro indicam $\neg$*chuva*. (b) $\neg$*guarda-chuva* é observado em $t + 1$.Cada amostra é ponderada por sua probabilidade referente à observação,como indicam os tamanhos dos círculos. (c) Um novo conjunto de 10 amostras é gerado por seleção aleatória ponderada do conjunto atual, resultando em duas amostras que indicam *chuva* e oito que indicam $\neg$*chuva*.

Podemos mostrar que esse algoritmo é consistente — fornece as probabilidades corretas à medida que N tende a infinito —, considerando-se o que acontece durante um ciclo de atualização. Vamos supor que a população de amostras comece com uma representação correta da mensagem para a frente $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$ no tempo t. Escrevendo $N(\mathbf{x}_t \mid \mathbf{e}_{1:t})$ para representar o número de amostras que ocupam o estado $\mathbf{x}_t$ depois das observações $\mathbf{e}_{1:t}$ terem sido processadas, temos:

$$N(\mathbf{x}_t \mid \mathbf{e}_{1:t})/N = P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \tag{15.23}$$

para N grande. Agora, propagamos cada amostra para a frente, realizando a amostragem das variáveis de estados em $t + 1$, dados os valores para a amostra em t. O número de amostras que alcançam o estado $\mathbf{x}_{t+1}$ de cada $\mathbf{x}_t$ é a probabilidade de transição multiplicada pela população de $\mathbf{x}_t$; consequentemente, o número total de amostras que alcançam $\mathbf{x}_{t+1}$ é:

$$N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) N(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \,.$$

Agora, ponderamos cada amostra por sua probabilidade para a evidência em $t + 1$. Uma amostra no estado $\mathbf{x}_{t+1}$ recebe peso $P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1})$. O peso total das amostras em $\mathbf{x}_{t+1}$ depois de se ver $\mathbf{e}_{t+1}$ é portanto

$$W(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t}) \,.$$

Em seguida, vamos à etapa de reamostragem. Tendo em vista que cada amostra é replicada com probabilidade proporcional ao seu peso, o número de amostras no estado $\mathbf{x}_{t+1}$ depois da reamostragem é proporcional ao peso total em $\mathbf{x}_{t+1}$ antes da reamostragem:

$$\begin{aligned}
N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1})/N &= \alpha W(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) \\
&= \alpha P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t}) \\
&= \alpha P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) N(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \\
&= \alpha N P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \quad \text{(por 15.23)} \\
&= \alpha' P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \\
&= P(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) \quad \text{(por 15.5).}
\end{aligned}$$

Por conseguinte, a população da amostra após um ciclo de atualização representa corretamente a mensagem para a frente no tempo $t + 1$.

Então, a filtragem de partículas é *consistente*; porém, ela é *eficiente*? Na prática, parece que a resposta é sim: a filtragem de partículas parece manter boa aproximação em relação à distribuição posterior verdadeira com o uso de um número constante de amostras. Sob certas suposições — em particular, que as probabilidades nos modelos de transição e de sensores são estritamente maiores que 0 e menores que 1 —, é possível provar que a aproximação mantém erro delimitado com alta probabilidade. Pelo lado prático, a gama de aplicações tem crescido, incluindo muitos campos da ciência e da engenharia; algumas referências são dadas ao final do capítulo.

15.6 MANUTENÇÃO E CONTROLE DE MUITOS OBJETOS

As seções anteriores consideraram — sem mencionar — problemas de avaliação de estado envolvendo um único objeto. Nesta seção, veremos o que acontece quando dois ou mais objetos geram observações. O que torna esse caso diferente da avaliação de estado simples é que há agora a possibilidade da *incerteza* sobre qual objeto gerou qual observação. Esse é o problema de **incerteza de identidade** da Seção 14.6.3, agora visto em um contexto temporal. Na literatura da teoria de controle, esse é o problema de **associação de dados**, isto é, o problema de associar dados de observação com os objetos que os geraram.

O problema de associação de dados foi estudado inicialmente no contexto de rastreamento de radar, onde são detectados os pulsos refletidos em intervalos de tempo fixos por uma antena de radar rotativa. Em cada período de tempo, podem aparecer na tela múltiplos pontos de luz pequenos que representam um objeto no radar, mas não há nenhuma observação direta de quais pontos no tempo t pertencem a quais pontos no tempo $t - 1$. A Figura 15.19(a) mostra um exemplo simples com dois pontos de luz por período de tempo de cinco etapas. Sejam as duas localizações do ponto de luz no tempo t e^1_t e e^2_t (a rotulagem dos pontos de luz em um intervalo de tempo como "1" e "2" é completamente arbitrária e não carrega informações). Suponhamos, por enquanto, que exatamente duas aeronaves, A e B, geraram os pontos de luz; suas posições verdadeiras são X^A_t e X^B_t. Para simplificar, vamos supor também que cada aeronave se mova independentemente de acordo com um modelo de transição conhecido, por exemplo, um modelo linear gaussiano, como o utilizado no filtro de Kalman (Seção 15.4).

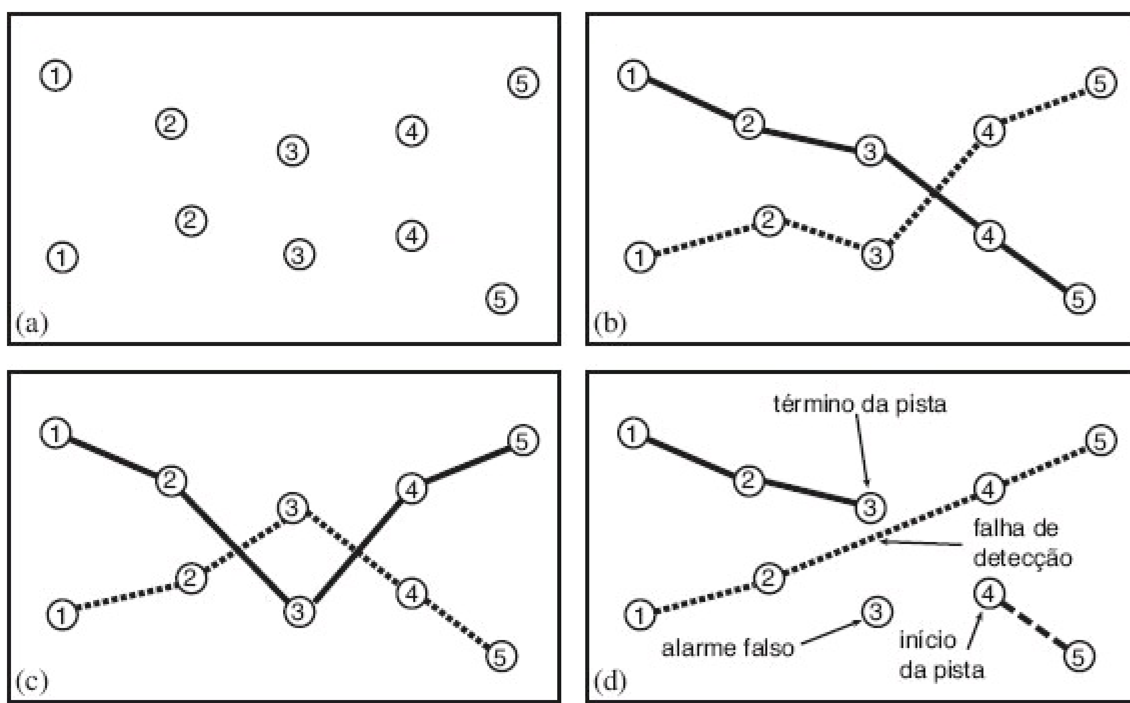


Figura 15.19 (a) Observações feitas a partir de localizações de objetos no espaço 2-D ao longo de cinco etapas de tempo. Cada observação é rotulada com o período de tempo, mas não identifica o objeto que o produziu. (b–c) Possíveis suposições sobre as pistas do objeto subjacente. (d) Uma suposição para o caso em que é possível: alarmes falsos, falha de detecção e inicio/término de pista.

Suponha que tentamos escrever o modelo de probabilidade completo para esse cenário, assim como fizemos para os processos temporais gerais na Equação 15.3. Como de costume, os fatores de distribuição conjuntos em contribuição para cada período de tempo são os seguintes:

$$P(x_{0:t}^A, x_{0:t}^B, e_{1:t}^1, e_{1:t}^2) = P(x_0^A)P(x_0^B)\prod_{i=1}^{t} P(x_i^A \mid x_{i-1}^A)P(x_i^B \mid x_{i-1}^B)\, P(e_i^1, e_i^2 \mid x_i^A, x_i^B)\,. \quad (15.24)$$

Gostaríamos de decompor o termo de observação $P(e_i^1, e_i^2 \mid x_i^A, x_i^B)$ em um produto de dois termos, um para cada objeto, mas isso exigiria saber que observação foi gerada por qual objeto. Em vez disso, temos que somar todas as formas possíveis de associar as observações com os objetos. Algumas dessas formas são mostradas na Figura 15.19 (b–c); em geral, para n objetos e T períodos de tempo, existem $(n!)^T$ maneiras de fazê-lo — um número muito grande.

Matematicamente falando, a “maneira de associar as observações com os objetos” é uma coleção de variáveis aleatórias não observadas que identificam a origem de cada observação. Vamos escrever ω_t para indicar o mapeamento um a um a partir de objetos para observações no tempo t, com $\omega_t(A)$ e $\omega_t(B)$ indicando as observações específicas (1 ou 2) que ω_t atribui a A e B (para n objetos, ω_t terá $n!$ valores possíveis; aqui, $n! = 2$). Porque os rótulos “1” e “2” sobre as observações são atribuídos arbitrariamente, *a priori* ω_t é uniforme sobre ω_t e independente dos estados dos objetos, x_t^A e x_t^B). Assim, podemos condicionar o termo de observação $P(e_i^1, e_i^2 \mid x_i^A, x_i^B)$ sobre ω_t e simplificar:

$$
\begin{aligned}
P(e_i^1, e_i^2 \mid x_i^A, x_i^B) &= \sum_{\omega_i} P(e_i^1, e_i^2 \mid x_i^A, x_i^B, \omega_i) P(\omega_i \mid x_i^A, x_i^B) \\
&= \sum_{\omega_i} P(e_i^{\omega_i(A)} \mid x_i^A) P(e_i^{\omega_i(B)} \mid x_i^B) P(\omega_i \mid x_i^A, x_i^B) \\
&= \frac{1}{2} \sum P(e_i^{\omega_i(A)} \mid x_i^A) P(e_i^{\omega_i(B)} \mid x_i^B) .
\end{aligned}
$$

Ligando com a Equação 15.24, obtemos uma expressão apenas em termos do modelo de transição e de sensores para objetos e observações individuais.

Assim como ocorre para todos os modelos de probabilidade, inferência significa somar sobre as variáveis que não sejam a consulta e a evidência. Para filtragem em MOMs e DBNs, fomos capazes de somar as variáveis de estado de 1 a $t - 1$ por um truque simples de programação dinâmica, para os filtros de Kalman, aproveitando propriedades especiais gaussianas. Para associação de dados, tivemos menos sorte. Não há algoritmo (conhecido) exato eficiente, pela mesma razão que não existe para a filtragem de comutação de Kalman: a distribuição de filtragem $P(x_t^A \mid e_{1:t}^1, e_{1:t}^2)$ para o objeto A acaba como uma mistura de muitas distribuições exponenciais, uma para cada forma de escolher uma sequência de observações para atribuir a A.

Como resultado da complexidade da inferência exata, foram utilizados muitos métodos aproximados diferentes. A abordagem mais simples é escolher uma única atribuição "melhor" em cada período de tempo, dadas as posições dos objetos previstas no período de tempo atual. Essa atribuição associa observações com objetos e permite que a pista de cada objeto seja atualizada e feita uma previsão para o próximo período de tempo. Para escolher a "melhor" atribuição, é comum usar o chamado **filtro de vizinho mais próximo**, que escolhe repetidamente o par de posições previsto e a observação mais próxima e adiciona esse par à atribuição. O filtro de vizinho mais próximo funciona bem quando os objetos estão bem separados no espaço de estados e o erro de previsão e incerteza é pequeno — em outras palavras, quando não há possibilidade de confusão. Quando há mais incerteza quanto à atribuição correta, uma abordagem melhor é escolher a atribuição que maximiza a probabilidade conjunta das observações atuais, dadas as posições previstas. Isso pode ser feito de forma muito eficiente usando o **algoritmo húngaro** (Kuhn, 1955), embora haja $n!$ atribuições para escolher.

Qualquer método que se compromete com uma atribuição única e melhor em cada período de tempo falha miseravelmente em condições mais difíceis. Em particular, se o algoritmo compromete-se com uma atribuição incorreta, a previsão no próximo período de tempo pode estar significativamente errada, conduzindo a mais atribuições incorretas, e assim por diante. Duas abordagens modernas acabam por ser muito mais eficazes. Um algoritmo de **filtragem de partícula** para associação de dados funciona pela manutenção de uma grande coleção de atribuições atuais possíveis. Um algoritmo CMMC explora o espaço histórico de atribuições — por exemplo, a Figura 15.19 (b−c) pode ser declarada no espaço de estados CMMC — e pode mudar de opinião sobre decisão de atribuições anteriores. Os métodos de dados de associação atuais, CMMC, podem lidar com muitas centenas de objetos em tempo real enquanto fornecem uma boa aproximação para as distribuições posteriores verdadeiras.

O cenário descrito até agora envolveu n objetos conhecidos gerando n observações em cada período de tempo. A aplicação real da associação de dados é tipicamente muito mais complicada.

Muitas vezes, as observações relatadas incluem alarmes falsos (também conhecidos como **sinal indesejado**), que não são provocados por objetos reais. Podem ocorrer **falhas de detecção**, o que significa que nenhuma observação é relatada com relação a um objeto real. Finalmente, chegam objetos novos e os antigos desaparecem. Esse fenômeno, que cria ainda mais mundos possíveis com os quais se preocupar, está ilustrado na Figura 15.19(d).

A Figura 15.20 mostra duas imagens de câmeras amplamente separadas em uma rodovia da Califórnia. Nessa aplicação, estamos interessados em dois objetivos: estimar o tempo que leva, em termos das condições de tráfego atual, para ir de um lugar para outro no sistema de autoestrada e medir a *demanda*, ou seja, quantos veículos rodam entre dois pontos quaisquer no sistema em determinado período de tempo do dia e em dias específicos da semana. Ambos os objetivos requerem a resolução do problema de associação de dados em uma área ampla com muitas câmeras e dezenas de milhares de veículos por hora. Com a fiscalização visual, sombras em movimento, veículos articulados, reflexos nas poças de água etc. causam alarmes falsos; oclusão, nevoeiro, escuridão e falta de contraste visual causam falhas de detecção; e os veículos estão constantemente entrando e saindo do sistema da rodovia. Além disso, o aparecimento de qualquer veículo pode mudar drasticamente entre as câmeras, dependendo das condições de iluminação e da posição do veículo na imagem, e o modelo de transição muda à medida que os engarrafamentos vêm e vão. Apesar desses problemas, os algoritmos de associação de dados modernos têm obtido sucesso ao estimar parâmetros de tráfego em contextos do mundo real.


(a)


(b)

Figura 15.20 Imagens da fiscalização por câmeras do fluxo de subida (a) e descida (b) cerca de duas milhas distante da Autoestrada 99, em Sacramento, Califórnia. O veículo marcado com contorno foi identificado nas duas câmeras.

As associações de dados são uma base essencial para manter o controle de um mundo complexo porque sem elas não há maneira de combinar múltiplas observações de determinado objeto. Quando os objetos do mundo interagem uns com os outros em atividades complexas, a compreensão do mundo requer a combinação de associação de dados com os modelos de probabilidade relacional e de universo aberto da Seção 14.6.3. Essa é atualmente uma área de pesquisa ativa.

15.7 RESUMO

Este capítulo tratou do problema geral de representar e raciocinar sobre processos temporais probabilísticos. Os principais pontos foram:

- O estado mutável do mundo é manipulado pela utilização de um conjunto de variáveis aleatórias para representar o estado em cada instante no tempo.
- As representações podem ser projetadas para satisfazer à **propriedade de Markov**, de forma que o futuro seja independente do passado, dado o presente. Combinado com a hipótese de que o processo é **estacionário** — isto é, as dinâmicas não mudam ao longo do tempo —, isso simplifica bastante a representação.
- Um modelo de probabilidade temporal pode ser considerado a união de um **modelo de transição** que descreve a evolução e um **modelo de sensores** que descreve o processo de observação.
- As principais tarefas de inferência em modelos temporais são **filtragem**, **previsão**, **suavização** e cálculo da **explicação mais provável**. Cada uma dessas tarefas pode ser realizada com o emprego de algoritmos recursivos simples cujo tempo de execução é linear na duração da sequência.
- Três famílias de modelos temporais foram estudadas em maior profundidade: **modelos ocultos de Markov**, **filtros de Kalman** e **redes bayesianas dinâmicas** (que incluem os outros dois como casos especiais).
- A menos que sejam feitas suposições especiais, como em filtros de Kalman, a inferência exata com muitas variáveis de estados parece ser intratável. Na prática, o algoritmo de **filtragem de partículas** parece ser um algoritmo de aproximação efetivo.
- Ao tentar manter o controle de muitos objetos, surge a incerteza de quais observações pertencem a quais objetos — o problema de **associação de dados**. O número de hipóteses de associação é tipicamente bem intratável, mas os algoritmos CMMC e de filtragem de partículas para associação de dados funcionam bem na prática.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Muitas das ideias básicas para avaliação do estado de sistemas dinâmicos vieram do matemático C. F. Gauss (1809), que formulou um algoritmo determinístico de mínimos quadrados para o problema de estimar órbitas a partir de observações astronômicas. O matemático russo A. A. Markov (1913) desenvolveu aquilo que se denominou mais tarde **hipótese de Markov** em sua análise dos processos estocásticos; ele estimou uma cadeia de Markov de primeira ordem sobre letras do texto de *Eugene Onegin*. Levin *et al.* (2008) abordam a teoria geral da cadeia de Markov e a mistura de tempos.

Um trabalho secreto significativo sobre filtragem foi realizado durante a Segunda Guerra Mundial por Wiener (1942) para processos de tempo contínuo e por Kolmogorov (1941) para processos de tempo discreto. Embora esse trabalho tenha levado a importantes desenvolvimentos tecnológicos durante os 20 anos seguintes, seu uso de uma representação no domínio das frequências tornou muitos cálculos bastante incômodos. A modelagem direta do espaço de estados do processo estocástico acabou se mostrando mais simples, conforme demonstraram Peter Swerling (1959) e Rudolf Kalman (1960). Este último ensaio introduziu o que se conhece agora como filtro de Kalman para inferência

para a frente em sistemas lineares com ruído gaussiano; os resultados de Kalman, no entanto, foram previamente obtidos pelo estatístico dinamarquês Thorvold Thiele (1880) e pelo matemático russo Ruslan Stratonovich (1959), que Kalman encontrou em Moscou em 1960. Após uma visita à Nasa Ames Research Center, em 1960, Kalman viu a aplicabilidade do método de rastreamento de trajetórias de foguetes, e posteriormente o filtro foi implementado pelas missões Apollo. Importantes resultados em suavização foram derivados por Rauch *et al*. (1965), e o suavizador com o impressionante nome Rauch-Tung-Striebel ainda é uma técnica-padrão atualmente. Muitos resultados iniciais estão reunidos em Gelb (1974). Bar-Shalom e Fortmann (1988) apresentam tratamento mais moderno com uma abordagem de Bayes, além de muitas referências à vasta literatura sobre o assunto. Chatfield (1989) e Box *et al.* (1994) cobrem a abordagem da teoria de controle para análise de séries temporais.

O modelo oculto de Markov e os algoritmos associados para inferência e aprendizado, incluindo o algoritmo para a frente-para trás, foram desenvolvidos por Baum e Petrie (1966). O algoritmo de Viterbi apareceu primeiro em Viterbi (1967). Ideias semelhantes também apareceram independentemente na comunidade de filtragem de Kalman (Rauch *et al*., 1965). O algoritmo para a frente-para trás foi um dos principais precursores da formulação geral do algoritmo EM (Dempster *et al*., 1977); veja também o Capítulo 20. A suavização do espaço de constantes aparece em Binder *et al*. (1997b), como também o algoritmo de dividir e conquistar desenvolvido no Exercício 15.3. A suavização de retardo fixo de tempo constante para MOMs apareceu pela primeira vez em Russell e Norvig (2003). Foram encontradas muitas aplicações para MOMs no processamento da linguagem (Charniak, 1993), reconhecimento da fala (Rabiner e Juang, 1993), tradução automática (Och e Ney, 2003), biologia computacional (Krogh *et al*., 1994; Baldi *et al*., 1994), economia financeira (Bhar e Hamori, 2004) e outros campos. Houve várias extensões para o modelo MOM básico, por exemplo, o MOM hierárquico (Fine *et al*., 1998), e o MOM em camadas (Oliver *et al*., 2004) introduziu a estrutura de volta ao modelo, substituindo a variável de estado único dos MOMs.

As redes bayesianas dinâmicas (DBNs) podem ser visualizadas como uma codificação esparsa de um processo de Markov e foram utilizadas primeiro em IA por Dean e Kanazawa (1989b), Nicholson e Brady (1992) e Kjaerulff (1992). O último trabalho estende o sistema de rede de Hugin Bayes para acomodar redes bayesianas dinâmicas. O livro de Dean e Wellman (1991) ajudou a popularizar DBNs e a abordagem probabilística para o planejamento e controle dentro da IA. Murphy (2002) forneceu uma análise completa de DBNs.

As redes bayesianas dinâmicas se tornaram populares para modelar uma variedade de processos de movimentos complexos em visão de computadores (Huang *et al*., 1994; Intille e Bobick, 1999). Como MOMs, encontraram aplicação em reconhecimento de fala (Zweig e Russell, 1998; Richardson *et al*., 2000; Stephenson *et al*., 2000; Nefian *et al*., 2002; Livescu *et al*., 2003), genômica (Murphy e Mian, 1999; Perrin *et al*., 2003; Husmeier, 2003) e localização robótica (Theocharous *et al*., 2004). A ligação entre MOMs e DBNs, e entre algoritmo para a frente e para trás e propagação de rede bayesiana, foi feita explicitamente por Smyth *et al.* (1997). Uma unificação adicional com filtros de Kalman (e outros modelos estatísticos) aparece em Roweis e Ghahramani (1999). Existem procedimentos para aprender os parâmetros (Binder *et al*., 1997a; Ghahramani, 1998) e as estruturas (Friedman *et al*., 1998) de DBNs.

O algoritmo de filtragem de partículas descrito na Seção 15.5 tem uma história particularmente

interessante. Os primeiros algoritmos de amostragem para filtragem de partículas (também chamado de método sequencial de Monte Carlo) foram desenvolvidos na comunidade de teoria de controle por Handschin e Mayne (1969), e a ideia de reamostragem que constitui o núcleo da filtragem de partículas apareceu em um periódico russo sobre controle (Zaritskii *et al*., 1975). Mais tarde, ele foi recriado em estatística como reamostragem sequencial do tipo **amostragem de importância**, ou **SIR** (Rubin, 1988; Liu e Chen, 1998), em teoria de controle como filtragem de partículas (Gordon *et al*., 1993; Gordon, 1994), em IA como **sobrevivência do mais adaptado** (Kanazawa *et al*., 1995) e em visão de computadores como **condensação** (Isard e Blake, 1996). O artigo de Kanazawa *et al*. (1995) inclui um aperfeiçoamento denominado **reversão da evidência**, pelo qual o estado no tempo $t + 1$ tem uma amostragem condicional sobre o estado no tempo *t e sobre a evidência no tempo* $t + 1$. Isso permite que a evidência influencie diretamente a geração de amostras, e foi demonstrado por Doucet (1997) e Liu e Chen (1998) para reduzir o erro de aproximação. A filtragem de partícula foi aplicada em muitas áreas, incluindo o rastreamento de padrões de movimento complexos em vídeo (Isard e Blake, 1996), a previsão do mercado de ações (Freitas *et al*., 2000), e diagnóstico de falhas em veículos planetários (Verma *et al*., 2004). Uma variante chamada de **filtro de partículas Rao-Blackwellized** ou filtro de partículas FPRB (Doucet *et al*., 2000;. Murphy e Russell, 2001) aplica filtragem de partícula em um subconjunto de variáveis de estado e, para cada partícula, realiza inferência exata sobre as demais variáveis condicionadas à sequência de valor na partícula. Em alguns casos, o FPRB funciona bem com milhares de variáveis de estado. No Capítulo 25 será descrita uma aplicação de FPRB para localização e mapeamento em robótica. O livro de Doucet *et al*. (2001) junta muitos artigos importantes sobre o algoritmo **sequencial de Monte Carlo** (SMC), do qual a filtragem de partículas é o exemplo mais importante. Pierre Del Moral e seus colegas realizaram extensas análises teóricas dos algoritmos de SMC (Del Moral, 2004; Del Moral *et al*., 2006).

Os métodos CMMC (veja a Seção 14.5.2) podem ser aplicados para problemas de filtragem, por exemplo, a amostragem de Gibbs pode ser aplicada diretamente a uma DBN desenrolada. Para evitar o problema de aumento nos tempos de atualização à medida que a rede cresce, a filtragem **CMMC decomposta** (Marthi *et al*., 2002) prefere amostrar variáveis de estado mais recentes, com probabilidade de que se decomponha de $1/k^2$ para a variável k períodos para o passado. O CMMC decomposto é um filtro provavelmente não divergente. Os teoremas não divergentes podem também ser obtidos de certos tipos de **filtragem de densidade assumida**.

Um filtro de densidade assumida pressupõe que a distribuição *a posteriori* sobre os estados no tempo t pertence a determinada família finitamente parametrizada; se as etapas de projeção e atualização a levarem para fora dessa família, a distribuição será projetada para trás para dar a melhor aproximação dentro da família. Para DBNs, o algoritmo de Boyen-Koller (Boyen *et al*., 1999) e o algoritmo de **fronteira fatorada** (Murphy e Weiss, 2001) assumem que a distribuição posterior pode ser bem aproximada por um produto de fatores pequenos. Técnicas variacionais (veja o Capítulo 14) também foram desenvolvidas para modelos temporais. Ghahramani e Jordan (1997) descrevem um algoritmo de aproximação para o **MOM fatorial**, uma DBN na qual duas ou mais cadeias de Markov que evoluem de modo independente são vinculadas por um fluxo de observação compartilhada. Jordan *et al*. (1998) focalizam várias outras aplicações.

A associação de dados para rastrear alvos múltiplos foi descrita pela primeira vez em um cenário

probabilístico por Sittler (1964). O primeiro algoritmo prático para problemas de larga escala era o "rastreador de múltiplas hipóteses" ou RMH (Reid, 1979). Bar-Shalom e Fortmann (1988) e Bar-Shalom (1992) juntaram muitos artigos importantes. Deve-se a Pasula *et al*. (1999) o desenvolvimento de um algoritmo CMMC para associação de dados, que aplicou para os problemas de fiscalização de tráfego. Oh *et al*. (2009) fornecem uma análise formal e comparações experimentais extensas a outros métodos. Schulz *et al*. (2003) descrevem um método de associação de dados com base em filtragem de partícula. Ingemar Cox analisou a complexidade de associação de dados (Cox, 1993; Cox e Hingorani, 1994) e trouxe o assunto à atenção da visão comunitária. Observou também a aplicabilidade do algoritmo húngaro de tempo polinomial para o problema de encontrar atribuições mais prováveis, que havia sido considerado por muito tempo um problema intratável na comunidade de rastreamento. O algoritmo em si foi publicado por Kuhn (1955), com base em traduções de artigos publicados em 1931 por dois matemáticos húngaros, Dénes König e Jenö Egerváry. No entanto, o teorema fundamental havia sido derivado anteriormente de um manuscrito inédito em latim pelo famoso matemático prussiano Carl Gustav Jacobi (1804-1851).

EXERCÍCIOS

15.1 Mostre que qualquer processo de Markov de segunda ordem pode ser reescrito como um processo de Markov de primeira ordem com um conjunto ampliado de variáveis de estados. Isso sempre pode ser feito de maneira *parcimoniosa*, isto é, sem aumentar o número de parâmetros necessários para especificar o modelo de transição?

15.2 Neste exercício, examinamos o que acontece com probabilidades no mundo do guarda-chuva, no limite de longas sequências de tempo.

a. Suponha que observamos uma sequência interminável de dias em que o guarda-chuva aparece. Mostre que, à medida que passam os dias, a probabilidade de chuva no dia atual aumenta monotonicamente, tendendo a um ponto fixo. Calcule esse ponto fixo.

b. Agora, considere uma *previsão* cada vez mais longe no futuro, dadas apenas as duas primeiras observações de guarda-chuva. Primeiro, calcule a probabilidade $P(r_{2+\mathrm{k}} \mid u_1, u_2)$ para $k = 1 \ldots 20$ e represente os resultados em um gráfico. Você deverá verificar que a probabilidade converge em direção a um ponto fixo. Demonstre que o valor exato desse ponto fixo é 0,05.

15.3 Este exercício desenvolve uma variante com eficiência de espaço do algoritmo para a frente-para trás descrito na Figura 15.4. Desejamos calcular $\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t})$ para $k = 1, \ldots, t$. Isso será feito com uma abordagem de dividir e conquistar.

a. Suponha, por simplicidade, que t seja ímpar, e seja $h = (t + 1)/2$ o ponto médio. Mostre que $\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t})$ pode ser calculada para $k = 1, \ldots, h$, dada apenas a mensagem para a frente inicial $\mathbf{f}_{1:0}$, a mensagem para trás $\mathbf{b}_{h+1:t}$ e a evidência $\mathbf{e}_{1:h}$.

b. Mostre um resultado semelhante para a segunda metade da sequência.

c. Dados os resultados de (a) e (b), é possível construir um algoritmo de dividir e conquistar, executando-se o algoritmo primeiro no sentido para a frente ao longo da sequência e depois no

sentido para trás a partir do fim, armazenando apenas as mensagens exigidas no ponto médio e nas extremidades. Em seguida, o algoritmo é chamado em cada metade. Escreva o algoritmo em detalhes.

d. Calcule a complexidade de tempo e espaço do algoritmo como uma função de t, a duração da sequência. Como isso se altera se dividirmos a entrada em mais de dois fragmentos?

15.4 Neste capítulo, delineamos um procedimento com falhas para descobrir a sequência de estados mais provável, dada uma sequência de observações. O procedimento envolve a descoberta do estado mais provável em cada período de tempo, o uso da suavização e o retorno da sequência composta por esses estados. Mostre que, para alguns modelos de probabilidade temporal e sequências de observações, esse procedimento retorna uma sequência de estados impossível (isto é, a probabilidade posterior da sequência é zero).

15.5 A Equação 15.12 descreve o processo de filtragem para a formulação da matriz do MOMs. Forneça uma equação semelhante para o cálculo de probabilidades, que foi descrita genericamente na Equação 15.7.

15.6 Considere o mundo do aspirador de pó da Figura 4.18 (sensoriamento perfeito) e a Figura 15.7 (sensoriamento ruidoso). Suponha que o robô receba uma sequência de observações, tais que, com o sensoriamento perfeito, há apenas uma localização perfeita em que ele poderia estar. Será essa a localização necessariamente mais provável para a probabilidade ϵ de ruído suficientemente pequeno sob sensoriamento ruidoso? Prove sua argumentação ou forneça um contra exemplo.

15.7 Na Seção 15.3.2, a distribuição prévia sobre as localizações é uniforme e o modelo de transição assume a mesma probabilidade de movimento para qualquer quadrado vizinho. E se esses pressupostos estiverem errados? Suponha que a localização inicial seja realmente escolhida de forma uniforme a partir do quadrante noroeste da sala e a ação *Mover* realmente tenda a mover para o sudeste. Mantendo o modelo MOM fixo, explore o efeito sobre a precisão da localização e do caminho à medida que aumenta a tendência para o sudeste, para diferentes valores de ϵ.

15.8 Considere uma versão do robô do aspirador de pó cujo programa de ação é ir em linha reta pelo tempo que puder; só quando encontra um obstáculo ele muda para uma nova direção (selecionada aleatoriamente). Para modelar esse robô, cada estado no modelo consiste em um par (*localização*, *direção*). Implemente esse modelo e veja quão bem o algoritmo de Viterbi pode acompanhar um robô com esse modelo. Por ser o programa de ação do robô mais restrito do que o do robô de passeio aleatório, significa que as previsões do caminho mais provável são mais precisas?

15.9 Este exercício trata da filtragem em um ambiente sem pontos de referência. Considere um robô de aspirador de pó em uma sala vazia, representado por uma grade retangular $n \times$ m. A localização do robô está oculta; a única evidência disponível para o observador é um sensor de localização ruidoso que dá uma aproximação da localização do robô. Se o robô estiver na posição (x, y), com probabilidade 0,1 o sensor indica a localização correta, com probabilidade 0,05 ele relata um dos oito locais circundantes (x, y), com probabilidade 0,025 ele relata uma das 16 localidades que rodeiam os oito, e com a probabilidade remanescente de 0,1 ele relata “sem leitura”. O programa de ação do robô é escolher uma direção e segui-la com probabilidade 0,8 em cada etapa; o robô muda para uma nova direção selecionada aleatoriamente com probabilidade 0,2 (ou com probabilidade 1

caso encontre uma parede). Implemente isso como um MOM e faça a filtragem para acompanhar o robô. Com que precisão podemos rastrear o caminho do robô?

15.10 Com frequência, desejamos monitorar um sistema de estados contínuos cujo comportamento se alterna de maneira imprevisível entre um conjunto de k "modos" distintos. Por exemplo, uma aeronave que tenta escapar de um míssil pode executar uma série de manobras distintas que o míssil talvez tente acompanhar. Uma representação de rede bayesiana de tal modelo de **filtro de Kalman de comutação** é mostrada na Figura 15.21.

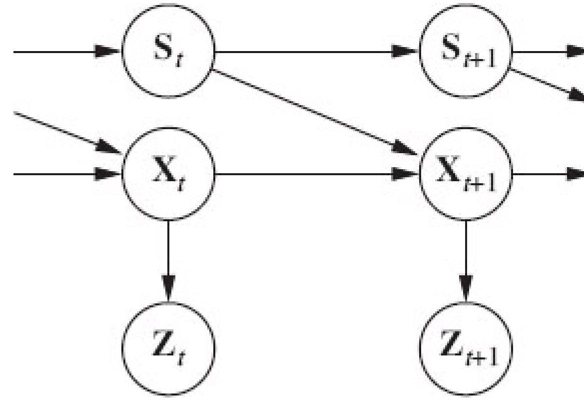


Figura 15.21 Representação de rede bayesiana de um filtro de Kalman de comutação. A variável de comutação S_t é uma variável de estado discreto cujo valor determina o modelo de transição para as variáveis de estados contínuos $\mathbf{X}_t$. Para qualquer estado discreto i, o modelo de transição $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, S_t = i)$ é um modelo gaussiano linear, da mesma maneira que ocorre em um filtro de Kalman normal. O modelo de transição para o estado discreto, $\mathbf{P}(S_{t+1} \mid S_t)$, pode ser considerado uma matriz, como ocorre em um modelo oculto de Markov.

a. Suponha que o estado discreto *St* tenha k valores possíveis e que a estimativa de estado anterior contínuo $\mathbf{P}(\mathbf{X}_0)$ seja uma distribuição gaussiana multivariada. Mostre que a previsão $\mathbf{P}(\mathbf{X}_1)$ é uma **mistura de gaussianos**, isto é, uma soma ponderada de gaussianos, tal que a soma dos pesos seja igual a 1.

b. Mostre que, se a estimativa atual de estados contínuos $\mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$ é uma mistura de m gaussianos, então no caso geral a estimativa atualizada de estado $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1})$ será uma mistura de km gaussianos.

c. Que aspecto do processo temporal os pesos na mistura gaussiana representam?

Juntos, os resultados de (a) e (b) mostram que a representação da distribuição posterior cresce sem limite, até mesmo para filtros de Kalman de comutação, que são os modelos dinâmicos híbridos mais simples.

15.11 Complete a etapa omitida na derivação da Equação 15.19, a primeira etapa de atualização para o filtro de Kalman unidimensional.

15.12 Vamos examinar o comportamento da atualização de variância na Equação 15.20.

a. Represente o valor de σ_t^2 como uma função de t, dados diversos valores para σ_x^2 e σ_z^2.

b. Mostre que a atualização tem um ponto fixo σ^2 tal que $\sigma_t^2 \rightarrow \sigma^2$ à medida que $t \rightarrow \infty$, e calcule o

valor de σ^2.

c. Forneça uma explicação qualitativa para o que acontece à medida que $\sigma_x^2 \to 0$ e $\sigma_z^2 \to 0$.

15.13 Um professor quer saber se os estudantes estão dormindo o suficiente. Cada dia, o professor observa se os alunos dormem em sala de aula e se têm os olhos vermelhos. O professor tem a seguinte teoria de domínio:

- A probabilidade mais forte de dormir o suficiente, sem observações, é de 0,7.
- A probabilidade de dormir o suficiente na noite t é de 0,8, dado que o aluno dormiu o suficiente na noite anterior, e 0,3 se não dormiu.
- A probabilidade de ter os olhos vermelhos é de 0,2 se o aluno dormiu o suficiente e 0,7 se não dormiu.
- A probabilidade de dormir em sala de aula é de 0,1 se o aluno dormiu o suficiente e 0,3 se não dormiu.

Formule essa informação como uma rede bayesiana dinâmica que o professor poderia utilizar para filtrar ou prever a partir de uma sequência de observações. Em seguida, reformule-a como um modelo oculto de Markov que tem apenas uma variável única de observação. Forneça as tabelas de probabilidade completas para o modelo.

15.14 Para a DBN especificada no Exercício 15.13 e para os valores de evidência

$\mathbf{e}_1$ = sem olhos vermelhos, não dormir em sala de aula

$\mathbf{e}_2$ = olhos vermelhos, não dormir em sala de aula

$\mathbf{e}_3$ = olhos vermelhos, dormir em sala de aula

realizar os seguintes cálculos:

a. Estimativa de estado: calcule $P(SonoSuficiente_t \mid e_{1:t})$ para cada um dos $t = 1, 2, 3$.

b. Suavização: calcule $P(SonoSuficiente_t \mid e_{1:3})$ para cada um dos $t = 1, 2, 3$.

c. Compare as probabilidades suavizadas e filtradas para $t = 1$ e $t = 2$.

15.15 Suponha que um aluno em particular apareça com os olhos vermelhos e durma na sala de aula todos os dias. Dado o modelo descrito no Exercício 15.13, explique por que a probabilidade que o aluno tenha dormido o suficiente na noite anterior converge para um ponto fixo em vez de continuar a descer enquanto reunimos mais dias de evidência. Qual é o ponto fixo? Responda tanto numericamente (por cálculo) como analiticamente.

15.16 Este exercício analisa em mais detalhes o modelo de falha persistente correspondente ao sensor de bateria da Figura 15.15(a).

a. A Figura 15.15(b) para em $t = 32$. Descreva qualitativamente o que deve acontecer à medida que $t \to \infty$ caso a leitura do sensor continue a ser 0.

b. Suponha que a temperatura externa afete o sensor da bateria de tal modo que as falhas transientes se tornem mais prováveis à medida que a temperatura aumenta. Mostre como ampliar a estrutura de DBN da Figura 15.15(a) e explique quais são as mudanças necessárias nas TPCs.

c. Dada a nova estrutura de rede, as leituras do nível de carga da bateria podem ser utilizadas pelo robô para deduzir a temperatura atual?

15.17 Considere a aplicação do algoritmo de eliminação de variáveis à DBN de guarda-chuva desenvolvida para três fatias, onde a consulta é $\mathbf{P}(R_3 \mid u_1, u_2, u_3)$. Mostre que a complexidade de espaço do algoritmo — o tamanho do maior fator — é a mesma, independentemente do fato de as variáveis de chuva serem eliminadas em ordem para a frente ou para trás.

---

[1] A incerteza sobre o tempo contínuo pode ser modelada por **equações diferenciais estocásticas** (EDSs). Os modelos estudados neste capítulo podem ser vistos como aproximações em tempo discreto para EDSs.

[2] O termo "filtragem" refere-se às raízes do problema no trabalho inicial de processamento de sinal, onde o problema era filtrar o ruído em um sinal estimando suas propriedades subjacentes.

[3] Em particular, ao acompanhar um objeto em movimento com observações de posição imprecisa, a suavização fornece uma trajetória estimada mais suave que a filtragem — daí o nome.

[4] O leitor pouco familiarizado com operações básicas sobre vetores e matrizes talvez deseje consultar o Apêndice A antes de prosseguir com o estudo desta seção.

[5] A rigor, uma distribuição gaussiana é problemática porque atribui probabilidade diferente de zero a níveis muito negativos de carga. Às vezes, a **distribuição beta** é uma escolha melhor para uma variável cujo intervalo é restrito.

CAPÍTULO 16

Tomada de decisões simples

Em que vemos como um agente deve tomar decisões de forma a obter o que deseja – pelo menos em média.

Neste capítulo, complementaremos os detalhes de como a teoria da utilidade combina com a teoria da probabilidade para formar um agente de teoria da decisão — um agente que pode tomar decisões racionais baseadas em suas crenças e no que ele deseja. Tal agente pode tomar decisões em contextos nos quais a incerteza e objetivos conflitantes deixam um agente lógico sem meios para se decidir. Na realidade, um agente baseado em objetivos faz distinção binária entre estados bons (objetivos) e estados ruins (não objetivos), enquanto um agente de teoria da decisão tem uma medida contínua da qualidade dos estados.

A Seção 16.1 introduz o princípio básico da teoria da decisão: a maximização da utilidade esperada. A Seção 16.2 mostra que o comportamento de qualquer agente racional pode ser captado supondo-se uma função utilidade que está sendo maximizada. A Seção 16.3 discute a natureza das funções utilidade com mais detalhes e, em particular, sua relação com quantidades individuais como o dinheiro. A Seção 16.4 mostra como tratar funções utilidade que dependem de diversas quantidades. Na Seção 16.5, descrevemos a implementação de sistemas de tomada de decisões. Em particular, introduzimos um formalismo chamado **redes de decisão** (também conhecido como **diagramas de influência**) que estende as redes bayesianas incorporando ações e utilidades. O restante do capítulo discute questões que surgem em aplicações da teoria da decisão a sistemas especialistas.

16.1 COMBINAÇÃO DE CRENÇAS E DESEJOS SOB INCERTEZA

A teoria da decisão, na sua forma mais simples, trata de escolher entre as ações com base na conveniência dos seus resultados *imediatos*, isto é, assume-se que o ambiente é episódico no sentido já definido (essa suposição será descrita no Capítulo 17). No Capítulo 3 utilizamos a notação RESULTADO(s_0, a) para o estado que é o resultado determinístico de tomar a ação a no estado s_0. Neste capítulo, trataremos de ambientes não determinísticos parcialmente observáveis. Como o agente pode não saber o estado atual, nós o omitimos e definimos RESULTADO(a) como uma *variável* aleatória cujos valores são os estados resultantes possíveis. A probabilidade do resultado

s', dadas as observações de evidências **e**, é escrita como

$$P(\text{RESULTADO}(a) = s' \mid a, \mathbf{e}),$$

onde o a no lado direito da barra significa o evento em que a ação a é executada.[1]

As preferências dos agentes são apreendidas por uma **função utilidade**, $U(s)$, que atribui um único número utilidade para expressar a conveniência de um estado. A **utilidade esperada** de uma ação, dada a evidência, $UE(a \mid \mathbf{e})$, é apenas o valor da utilidade média ponderada dos resultados, pela probabilidade que o resultado ocorra:

$$UE(a|\mathbf{e}) = \sum_{s'} P(\text{RESULTADO}(a) = s' \mid a, \mathbf{e})\, U(s')\,.$$

O princípio da **utilidade máxima esperada** (UME) diz que um agente racional deve escolher a ação que maximize a utilidade esperada do agente:

$$ação = \operatorname*{argmax}_{a} UE(a \mid \mathbf{e}).$$

De certo modo, o princípio de UME poderia ser visto como a definição de tudo em IA. Tudo o que um agente inteligente tem de fazer é calcular as diversas quantidades, maximizar a utilidade sobre suas ações e ir em frente. Porém, isso não significa que o problema da IA seja *resolvido* pela definição!

O princípio UME *formaliza* a noção geral de que o agente deve "fazer a coisa certa", mas percorre apenas uma pequena distância em direção a uma *operacionalização* desse conselho. Estimar o estado do mundo exige percepção, aprendizagem, representação do conhecimento e inferência. O cálculo $P(\text{RESULTADO}(a) \mid a, \mathbf{e})$ requer um modelo causal completo do mundo e, como vimos no Capítulo 14, inferência NP-difícil em redes bayesianas (muito grandes). Calcular os resultados das utilidades $U(s')$ frequentemente requer pesquisa ou planejamento porque um agente pode não saber o quanto um estado é bom até que saiba onde pode chegar a partir desse estado. Assim, a teoria da decisão não é uma panaceia que resolve o problema de IA, mas fornece uma estrutura útil.

O princípio de UME tem relação clara com a ideia de medidas de desempenho introduzida no Capítulo 2. A ideia básica é muito simples. Considere os ambientes que poderiam levar um agente a ter um dado histórico de percepções e considere os diferentes agentes que poderíamos projetar. *Se um agente age para maximizar uma função utilidade que reflete corretamente a medida de desempenho pela qual seu comportamento está sendo julgado, ele alcançará a mais alta pontuação de desempenho possível (a média sobre todos os outros ambientes possíveis)*. Essa é a justificação central para o próprio princípio de UME. Embora a afirmação possa parecer tautológica, de fato ela incorpora uma transição muito importante a partir de um critério global externo de racionalidade — a medida de desempenho sobre históricos de ambientes até um critério local interno que envolve a maximização de uma função utilidade aplicada ao próximo estado.

16.2 A BASE DA TEORIA DA UTILIDADE

Intuitivamente, o princípio de utilidade máxima esperada (UME) parece um modo razoável de tomar decisões, mas não é de forma alguma evidente que ele seja o *único* modo racional. Afinal, por que maximizar a utilidade *média* é tão especial? O que há de errado com um agente que maximiza a soma ponderada dos cubos das utilidades possíveis ou tenta minimizar a pior perda possível? Além disso, um agente não poderia agir racionalmente apenas expressando preferências entre estados, sem lhes atribuir valores numéricos? Finalmente, por que deve existir uma função utilidade com as propriedades exigidas? Veremos.

16.2.1 Restrições sobre preferências racionais

Essas perguntas podem ser respondidas registrando-se algumas restrições sobre as preferências que um agente racional deve ter e depois mostrando-se que o princípio de UME pode ser derivado das restrições. Utilizamos a notação a seguir para descrever as preferências de um agente:

$A \succ B$ A é preferível a B.
$A \sim B$ O agente está indiferente entre A e B.
$A \succsim B$ O agente prefere A a B ou está indiferente entre eles.

Agora a pergunta óbvia é: que tipos de itens são A e B? Eles poderiam ser os estados do mundo, mas mais frequentemente do que nunca há incerteza sobre o que realmente está sendo oferecido. Por exemplo, o passageiro de uma companhia aérea a quem é oferecido um "prato de macarrão ou de frango" não sabe o que está sob o papel de alumínio.[2] A massa pode estar congelada ou deliciosa, o frango suculento ou cozido além do recomendável. Podemos pensar sobre o conjunto de resultados para cada ação como uma **loteria** — pense em cada ação como um bilhete. A notação da loteria L com os possíveis resultados $S_1,\ldots, S_n$ que ocorrem com probabilidades $p_1,\ldots, p_n$ é

$$L = [p_1, S_1; p_2, S_2;\ldots\ p_n, S_n].$$

Em geral, cada resultado S_i de uma loteria pode ser tanto um estado atômico como outra loteria. O principal problema da teoria da utilidade é entender como preferências entre loterias complexas estão relacionados com as preferências entre os estados subjacentes nessas loterias. Para resolver esse problema, listamos seis restrições que exigem qualquer preferência razoável a ser obedecida:

- **Ordenabilidade:** Dadas duas loterias quaisquer, um agente racional deve preferir uma à outra ou, então, classificar as duas como igualmente preferíveis.Ou seja, o agente não pode evitar a decisão. Como vimos anteriormente, recusar-se a apostar é como recusar-se a deixar o tempo passar.

  Apenas $(A \succ B)$, $(B \succ A)$, ou $(A \sim B)$ são possíveis.

- **Transitividade:** DadAs três loterias quaisquer, se um agente preferir A a B e preferir B a C,

então o agente deverá preferir A a C.

$$(A \succ B) \land (B \succ C) \Rightarrow (A \succ C).$$

- **Continuidade:** Se alguma loteria B estiver entre A e C em preferência, haverá alguma probabilidade p de que o agente racional fique indiferente entre escolher B por garantia ou escolher a loteria que produz A com probabilidade p e C com probabilidade $1 - p$.

$$A \succ B \succ C \Rightarrow \exists p\ [p, A; 1-p, C] \sim B.$$

- **Substitutibilidade:** Se um agente está indiferente entre duas loterias A e B, então o agente está indiferente entre duas outras loterias complexas que são a mesma loteria, exceto pelo fato de A ser substituído por B em uma delas. Isso é válido independentemente das probabilidades e do(s) outro(s) resultado(s) das loterias.

$$A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C].$$

Isto também é válido se substituirmos $\succ$ por ~ nesse axioma.

- **Monotonicidade:** Suponha que existam duas loterias que tenham os mesmos dois resultados, A e B. Se um agente prefere A a B, então o agente deve preferir a loteria que tem uma probabilidade mais alta para A (e vice-versa).

$$A \succ B \Rightarrow (p > q \Leftrightarrow [p, A; 1-p, B] \succ [q, A; 1-q, B]).$$

- **Decomponibilidade:** As loterias compostas podem ser reduzidas a loterias mais simples com o uso das leis da probabilidade. Isso se chama regra de “nada de diversão no jogo” porque afirma que duas loterias consecutivas podem ser compactadas em uma única loteria equivalente, como mostra a Figura 16.1(b).[3]

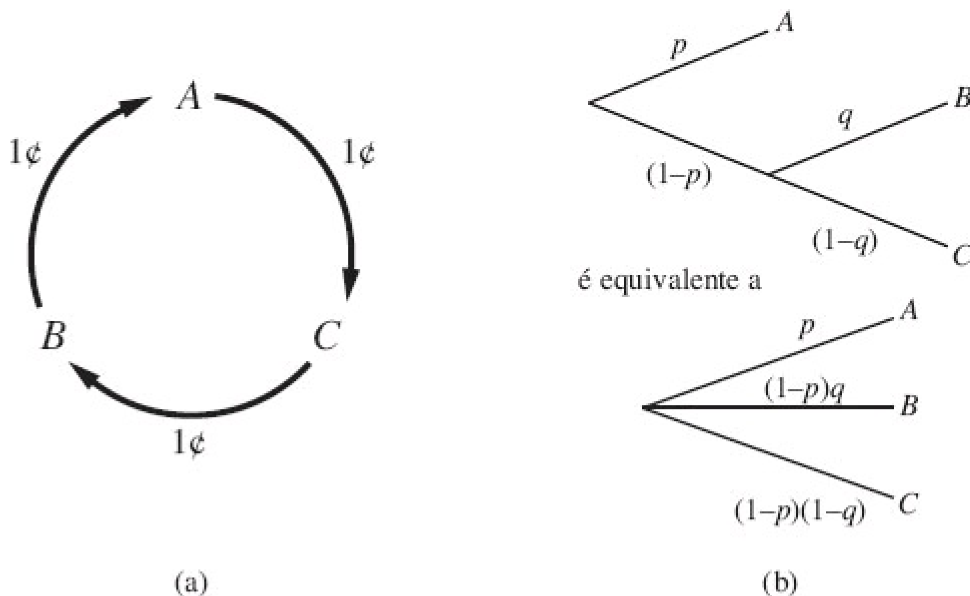


Figura 16.1 (a) Um ciclo de trocas mostrando que as preferências não transitivas $A \succ B \succ C \succ A$ resultam em comportamento irracional. (b) O axioma de decomponibilidade.

$$[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C].$$

Essas restrições são conhecidas como axiomas da teoria da utilidade. Cada axioma pode ser motivado mostrando que um agente que o viola vai exibir comportamento claramente irracional em algumas situações. Por exemplo, podemos motivar a transitividade fazendo com que um agente com preferências não transitivas nos dê todo o seu dinheiro. Suponha que o agente tenha as preferências intransitivas $A \succ B \succ C \succ A$ em que A, B e C são bens que podem ser trocados livremente. Se o agente tem A atualmente, poderemos oferecer a troca de C por A mais um centavo. O agente prefere C e, assim, estará disposto a fazer essa troca. Poderemos, então, oferecer a troca de B por C, tirando mais um centavo, e, finalmente, a troca de A por B. Isso nos leva de volta para onde começamos, com exceção que o agente nos deu três centavos (Figura 16.1(a)). Podemos continuar no ciclo todo até que o agente não tenha mais dinheiro. É claro que nesse caso o agente agiu irracionalmente.

16.2.2 Preferências levam à utilidade

Observe que os axiomas da teoria da utilidade são realmente axiomas sobre as preferências — eles não dizem nada sobre a função utilidade. Mas o fato é que dos axiomas de utilidade podemos derivar as seguintes consequências (para demonstração, consulte Von Neumann e Morgestern, 1944):

- **Existência da função utilidade:** Se as preferências de um agente obedecem aos axiomas de utilidade, existe uma função de valores reais U que opera sobre estados tais que $U(A) > U(B)$ se e somente se A é preferível em relação a B, e $U(A) = U(B)$ se e somente se o agente está indiferente entre A e B.

$$U(A) > U(B) \Leftrightarrow A \succ B$$
$$U(A) = U(B) \Leftrightarrow A \sim B.$$

- **Utilidade esperada de uma loteria:** A utilidade de uma loteria é o somatório da probabilidade de cada resultado vezes a utilidade desse resultado.

$$U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i U(S_i)\,.$$

Em outras palavras, uma vez que as probabilidades e as utilidades dos estados resultantes possíveis são especificadas, a utilidade de uma loteria composta envolvendo esses estados fica completamente determinada.

Como o resultado de uma ação não determinística é uma loteria, segue que um agente pode agir racionalmente — isto é, de forma consistente com suas preferências — somente pela escolha de uma ação que maximize a utilidade esperada de acordo com a Equação 16.1.

Os teoremas anteriores estabelecem que *existe* uma função utilidade para qualquer agente racional, mas eles não demonstram que ela é *única*. É fácil de ver, de fato, que o comportamento de um agente não mudaria se a sua função utilidade $U(S)$ fosse transformada de acordo com

$$U'(S) = aU(S) + b, \qquad (16.2)$$

onde a e b são constantes e $a > 0$; uma transformação afim.[4] Esse fato foi observado no Capítulo 5

para dois jogadores de jogos de azar; aqui, percebemos que é completamente genérico.

Como em um jogo, em um ambiente determinístico um agente só precisa de uma categorização de preferência dos estados — os números não importam. Isso é chamado de **função de valor** ou **função utilidade ordinal**.

É importante lembrar que a existência de uma função utilidade que descreve o comportamento de preferências de um agente não significa necessariamente que o agente esteja maximizando *explicitamente* essa função utilidade em suas próprias deliberações. Como mostramos no Capítulo 2, o comportamento racional pode ser gerado em qualquer número de aspectos. Porém, observando as preferências de um agente racional, um observador pode construir a função utilidade que representa o que o agente está de fato tentando realizar (mesmo se ele não souber disso).

16.3 FUNçõES UTILIDADE

A utilidade é uma função que faz o mapeamento de estados em números reais. Sabemos que existem alguns axiomas de utilidades aos quais todos os agentes racionais devem obedecer. Isso é tudo o que podemos dizer sobre as funções utilidade? No sentido exato, sim. Além das restrições listadas antes, um agente pode ter as preferências que desejar. Por exemplo, um agente pode preferir ter um número primo de reais em sua conta bancária; nesse caso, se tivesse $16, ele abriria mão de $3. Pode não ser usual, mas não podemos chamar de irracional. Um agente talvez preferisse um surrado Corcel 1973 a um brilhante Mercedes novo. As preferências também podem interagir: por exemplo, é possível que ele só prefira números primos de reais quando for proprietário do Corcel, mas quando tiver o Mercedes talvez prefira ter mais reais. Felizmente, as preferências de agentes reais em geral são mais sistemáticas e, por isso, mais fáceis de lidar.

16.3.1 Avalição de utilidade e escalas de utilidade

Se queremos construir um sistema de decisão teórica que ajude o agente a tomar decisões ou agir em seu nome, é preciso primeiro descobrir o que é a função utilidade do agente. Esse processo, muitas vezes chamado de **elicitação de preferência**, envolve a apresentação de escolhas para o agente e usa as preferências observadas para responder com precisão à função utilidade subjacente.

A Equação 16.2 informa que não existe escala absoluta de utilidades, mas é útil, no entanto, estabelecer uma *escala* em que as utilidades podem ser registradas e comparadas com qualquer problema particular. A escala pode ser estabelecida pela fixação das utilidades de quaisquer dois resultados particulares, da mesma forma que determinamos uma escala de temperatura, fixando o ponto de congelamento e o ponto de ebulição da água. Normalmente, fixamos a utilidade de um "melhor prêmio possível" em $U(S) = u_\top$ e da "pior catástrofe possível" em $U(S) = u_\perp$. **Utilidades normalizadas** usam uma escala com $u_\perp = 0$ e $u_\top = 1$.

Dada uma escala de utilidade entre $u_\top$ e $u_\perp$, podemos avaliar a utilidade de qualquer prêmio particular S pedindo para o agente escolher entre S e uma **loteria-padrão** $[p, u_\top; (1-p), u_\perp]$. A probabilidade p é ajustada até que o agente esteja indiferente entre S e a loteria-padrão. Assumindo

utilidades normalizadas, a utilidade S é dada por p. Uma vez feito isso, as loterias de todas as utilidades que envolvem aqueles prêmios são determinadas.

☞ Em problemas de decisão das áreas médica, de transporte e ambiental, entre outros, a vida da pessoas está em jogo. Em tais casos, $u_{\perp}$ é o valor atribuído à morte imediata (ou talvez a muitas mortes). *Embora ninguém se sinta confortável em definir um valor para a vida humana, o fato é que são estabelecidos compromissos o tempo todo*. As aeronaves recebem uma revisão completa a intervalos determinados pelos percursos e por milhas voadas, e não depois de cada viagem. Os carros são fabricados de uma forma que compense os custos das taxas de sobrevivência em um acidente. Paradoxalmente, uma recusa a "impor um valor monetário à vida" significa que a vida com frequência é *subestimada*. Ross Shachter relata experiência com uma agência governamental que subvencionou um estudo sobre a remoção do amianto nas escolas. Os analistas de decisão realizaram o estudo assumindo um valor em dólares em particular para a vida de uma criança em idade escolar, e argumentaram que a escolha racional sob essa suposição era remover o amianto. A agência governamental, moralmente afrontada, rejeitou o relatório. Em seguida, decidiu-se contra a remoção do amianto afirmando implicitamente um valor menor para a vida de uma criança do que o atribuído pelos analistas.

Algumas tentativas foram feitas para descobrir o valor que as pessoas dão à sua própria vida. "Moedas" comuns utilizadas em análise médica e de segurança são a **micromorte** (uma chance de morrer em um milhão) e o **QALY** (*quality-adjusted life year*), equivalente a um ano de boa saúde sem qualquer enfermidade. Se você perguntar às pessoas o quanto elas pagariam para evitar um risco — por exemplo, para evitar jogar roleta-russa com um revólver com milhões de balas — elas vão responder com números muito grandes, talvez dezenas de milhares de dólares, mas o seu comportamento real reflete um valor monetário muito mais baixo de uma micromorte. Por exemplo, dirigir um carro por 370 quilômetros incorre em um risco de uma micromorte; sobre a vida útil do seu carro, digamos 148.000 quilômetros, isto corresponde a 400 micromortes. As pessoas parecem estar dispostas a pagar a mais cerca de $10.000 (ao preço de 2009) por um seguro de carro que reduz pela metade o risco de morte ou cerca de $50 por micromorte. Muitos estudos confirmaram uma cifra nessa faixa entre muitos indivíduos e tipos de risco. Claro, esse argumento é válido apenas para riscos pequenos. A maioria das pessoas não concorda em se matar por $50 milhões.

Outra medida é o **QALY**, ou ano de vida ajustado pela qualidade. Pacientes com deficiência estão dispostos a aceitar menor expectativa de vida se a sua saúde integral for restaurada. Por exemplo, pacientes renais, em média, são indiferentes entre viver dois anos em uma máquina de diálise e um ano em plena saúde.

16.3.2 A utilidade do dinheiro

A teoria da utilidade tem suas raízes na economia, e a economia apresenta um candidato óbvio para se tornar uma medida de utilidade: o dinheiro (ou, mais especificamente, os bens líquidos totais de um agente). A quase universal capacidade de troca do dinheiro por todos os tipos de mercadorias e serviços sugere que o dinheiro desempenha um papel significativo nas funções humanas de utilidade.

Normalmente, o agente vai preferir mais dinheiro a menos dinheiro, sendo todos os outros itens iguais. Dizemos que o agente exiba uma **preferência monotônica** por mais dinheiro. No entanto, isso não significa que o dinheiro se comporta como função utilidade porque ele não diz nada sobre as preferências entre *loterias* que envolvem dinheiro.

Vamos supor que você tenha triunfado sobre os outros concorrentes em um programa de jogos pela televisão. Agora, o apresentador lhe oferece uma opção: levar o prêmio de $1.000.000 ou apostar tudo no lançamento de uma moeda (cara ou coroa). Se der cara, você acabará sem nada, mas, se der coroa, você ganhará US$2.500.000,00. Se for como a maioria das pessoas, você recusará o jogo e embolsará o milhão. Nesse caso, você estará sendo irracional?

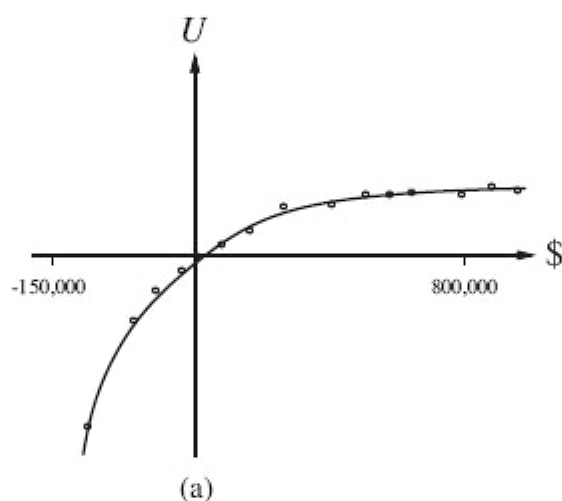
Supondo que você acredite que a moeda é justa, o **valor monetário esperado** (VME) do jogo é $\frac{1}{2}$ (\$0) + $\frac{1}{2}$(\$2.500.000) = \$1.250.000, que é mais que o prêmio original de \$1.000.000. Porém, isso não significa necessariamente que aceitar a aposta seja uma decisão melhor. Suponha que utilizamos Sn para denotar o estado de possuir a riqueza total \$$n$ e que sua riqueza atual seja \$$k$. Então, as utilidades esperadas das duas ações de aceitar e recusar o jogo são:

$$UE(Aceitar) = \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+2.500.000})$$
$$UE(Recusar) = U(S_{k+1.000.000})$$

Para determinar o que fazer precisamos atribuir utilidades aos estados resultantes. A utilidade não é diretamente proporcional ao valor monetário porque a utilidade para o seu primeiro milhão é muito alta (é o que achamos), enquanto a utilidade para um milhão adicional é menor.

Suponha que você atribua a utilidade 5 ao seu *status* financeiro atual (S_k), 9 ao estado $S_{k+2.500.000}$ e 8 ao estado $S_{k+1.000.000}$. Então, a ação racional seria recusar porque a utilidade esperada de aceitar é apenas 7 (menos que a utilidade 8 de recusar). Por outro lado, é mais provável que um bilionário tenha uma função utilidade que seja localmente linear no intervalo de poucos milhões a mais e, assim, aceitaria a aposta.

Em um estudo pioneiro das funções utilidade reais, Grayson (1960) descobriu que a utilidade do dinheiro era quase exatamente proporcional ao logaritmo da quantia (essa ideia foi sugerida primeiro por Bernoulli (1738); veja o Exercício 16.3). Uma curva específica, para um certo Mr. Beard, é mostrada na Figura 16.2(a). Os dados obtidos para as preferências de Mr. Beard são consistentes com uma função utilidade



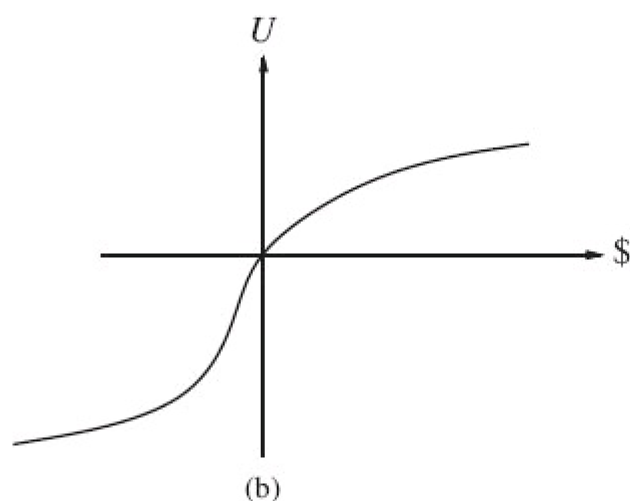


Figura 16.2 A utilidade do dinheiro. (a) Dados empíricos para Mr. Beard sobre um intervalo limitado. (b) Uma curva típica para o intervalo completo.

$$U(S_{k+n}) = -263{,}31 + 22{,}09\log(n + 150.000)$$

para o intervalo entre $n = -\$150.000$ e $n = \$800.000$.

Não devemos supor que essa seja a função utilidade definitiva para valor monetário, mas é provável que a maioria das pessoas tenha uma função utilidade côncava para riquezas positivas. Contrair dívidas é em geral considerado desastroso, mas as preferências entre diferentes níveis de dívidas podem exibir uma inversão da concavidade associada com riqueza positiva. Por exemplo, alguém que já deve $10.000.000 poderia muito bem aceitar uma aposta em um lançamento de moeda justo com um ganho de $10.000.000 para caras e uma perda de $20.000.000 para coroas.[5] Isso gera a curva em forma de *S* mostrada na Figura 16.2(b).

Vamos limitar nossa atenção à parte positiva das curvas, onde a declividade está diminuindo; então, para qualquer loteria *L*, a utilidade de se defrontar com essa loteria é menor que a utilidade de receber o valor monetário esperado da loteria como algo certo:

$$U(L) < U(S_{\text{VME}}(L))$$

Isto é, agentes com curvas dessa forma são **avessos ao risco**: eles preferem algo certo com compensação menor que o valor monetário esperado de uma aposta. Por outro lado, na região "desesperada" de grande riqueza negativa da Figura 16.2(b), o comportamento é de **busca do risco**.

O valor que um agente aceitará em vez de se arriscar em uma loteria é chamado **equivalente de certeza** da loteria. Os estudos mostram que a maioria das pessoas aceitará cerca de $400 em vez de uma aposta que ofereça $1.000 na metade do tempo e $0 na outra metade, ou seja, o equivalente de certeza da loteria é $400, enquanto o valor monetário esperado é $500. A diferença entre o valor monetário esperado de uma loteria e seu equivalente de certeza é chamado **prêmio de seguro**. A aversão ao risco é a base da indústria de seguros porque significa que os prêmios de seguros são positivos. As pessoas preferem pagar um prêmio de seguro pequeno a apostar o valor de sua casa contra a chance de um incêndio. Do ponto de vista da companhia de seguros, o preço da casa é muito pequeno comparado às reservas totais da firma. Isso significa que a curva de utilidade da seguradora é aproximadamente linear sobre essa pequena região, e o jogo não custa quase nada para a empresa.

Note que, no caso de *pequenas* mudanças de riqueza em relação à riqueza atual, quase qualquer curva será aproximadamente linear. Um agente que tenha uma curva linear é dito **neutro ao risco**. Portanto, no caso de apostas com pequenas somas, esperamos a neutralidade ao risco. De certo modo, isso justifica o procedimento simplificado que propôs pequenas apostas para avaliar as probabilidades e justificar os axiomas de probabilidade na Seção 13.2.3.

16.3.3 Utilidade esperada e decepção pós-decisão

A forma racional para escolher a melhor ação, a^*, é maximizar a utilidade esperada:

$$a^* = \underset{a}{\operatorname{argmax}}\, UE(a \mid \mathbf{e}).$$

Se tivermos calculado a utilidade esperada corretamente, de acordo com nosso modelo de probabilidade, e se o modelo probabilístico refletir corretamente os processos estocásticos subjacentes que geram os resultados, em média teremos a utilidade que esperamos se todo o processo for repetido muitas vezes.

Na realidade, porém, o nosso modelo geralmente simplifica demais a situação real, seja porque não sabemos o suficiente (por exemplo, ao fazer uma decisão de investimento complexo) seja porque o cálculo da verdadeira utilidade esperada é muito difícil (por exemplo, ao estimar a utilidade de estados sucessores do nó raiz no gamão). Nesse caso, estamos trabalhando realmente com estimativas $\widehat{UE}(a|\mathbf{e})$ da utilidade esperada verdadeira. Vamos supor, de forma cordial talvez, que as estimativas sejam **imparciais**, isto é, o valor esperado do erro, $E(\widehat{UE}(a|\mathbf{e}) - UE(a|\mathbf{e}))$, é zero. Nesse caso, ainda parece razoável escolher a ação com a mais alta utilidade estimada e esperar receber essa utilidade, em média, quando a ação for executada.

Infelizmente, o resultado real geralmente será significativamente *pior* do que estimamos, mesmo que a estimativa seja imparcial! Para saber o motivo, considere um problema de decisão em que haja k escolhas, cada uma com utilidade estimada verdadeira de 0. Suponha que o erro em cada estimativa de utilidade tenha zero de média e desvio-padrão de 1, como vemos na curva em negrito na Figura 16.3. Agora, como realmente começamos a gerar estimativas, alguns dos erros serão negativos (pessimistas) e outros positivos (otimistas). Como selecionamos a ação com a estimativa mais alta de utilidade, é óbvio que estaremos favorecendo as estimativas excessivamente otimistas, e isso é a origem da inclinação. Calcular a distribuição do máximo das estimativas k é uma questão simples (veja o Exercício 16.11) e, assim, temos como quantificar o grau de nossa decepção. A curva na Figura 16.3 para k = 3 tem média em torno de 0,85, de modo que a decepção média será cerca de 85% do desvio-padrão nas estimativas de utilidade.

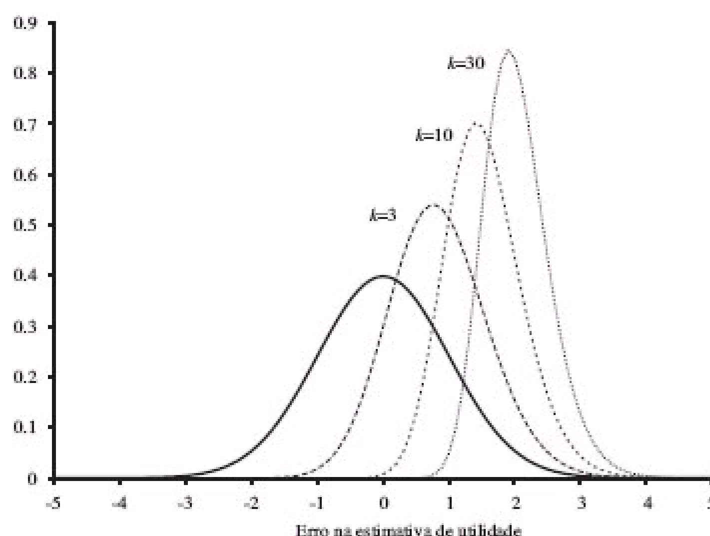


Figura 16.3 Representação gráfica do erro em cada uma das estimativas de utilidade k e da distribuição do máximo da estimativa para k = 3, 10 e 30.

Com mais opções, são mais propensas a surgirem estimativas extremamente otimistas: para k = 30, a decepção será de cerca de duas vezes o desvio-padrão das estimativas.

Essa tendência de a utilidade esperada estimada da melhor escolha ser demasiado elevada é chamada de **maldição do otimizador** (Smith e Winkler, 2006) e aflige mesmo os mais experientes analistas de decisão e estatísticos. Manifestações graves incluem acreditar que uma nova droga excelente que curou 80% dos doentes (foi escolhida entre k = milhares de drogas candidatas) ou o anúncio de um fundo mútuo teve retornos médios altos fará com que continue a tê-los (foi escolhido para aparecer no anúncio de k = dezenas de fundos na carteira total da companhia). Pode até mesmo ser o caso de o que parece ser a melhor escolha pode não ser, se a variação na estimativa de utilidade for alta: uma droga, selecionada a partir de milhares de tentativas, que curou 9 dentre 10 pacientes é provavelmente *pior* do que outra que tenha curado 800 dentre 1.000 pacientes.

A maldição do otimizador surge em todos os lugares por causa da onipresença dos processos de seleção da maximização de utilidade, portanto tomar as estimativas de utilidade pelo valor nominal é uma má ideia. Podemos evitar a maldição usando um modelo de probabilidade explícito $\mathbf{P}(\widehat{UE} \mid UE)$ do erro nas estimativas de utilidade. Dado esse modelo e um $\mathbf{P}(UE)$ anterior sobre o que poderíamos esperar razoavelmente que fossem as utilidades, lidamos com a estimativa de utilidade, uma vez obtida, como prova e calculamos a distribuição posterior para a utilidade verdadeira utilizando a regra de Bayes.

16.3.4 Julgamento humano e irracionalidade

A teoria da decisão é uma **teoria normativa**: ela descreve como um agente racional *deve* agir. A **teoria descritiva**, por outro lado, descreve como os agentes reais — por exemplo, os seres humanos — agem realmente. A aplicação da teoria econômica seria muito maior se os dois coincidissem, mas parece haver alguma evidência experimental em contrário. A evidência sugere que os seres humanos são "previsivelmente irracionais" (Ariely, 2009).

O problema mais conhecido é o paradoxo de Allais (Allais, 1953). As pessoas recebem uma escolha entre as loterias *A* e *B* e, em seguida, entre *C* e *D*, que têm os seguintes prêmios:

A: 80% de chance de $4.000
B: 100% de chance de $3.000
C: 20% de chance de $4.000
D: 25% de chance de $3.000.

A maioria das pessoas prefere consistentemente *B* sobre *A* (decidindo pelo certo) e *C* sobre *D* (considerando o maior VME). A análise normativa discorda! Podemos ver isso mais facilmente se usarmos a liberdade implícita pela Equação 16.2 para definir $U(\$0) = 0$. Nesse caso, $B \succ A$ implica que $U(\$3.000) > 0{,}8U(\$4.000)$, enquanto $C \succ D$ implica exatamente o inverso. Em outras palavras, não há nenhuma função utilidade que seja consistente com essas escolhas. Uma explicação para as preferências aparentemente irracionais é o **efeito certeza** (Kahneman e Tversky, 1979): as pessoas são fortemente atraídas para os ganhos que são certos. Existem várias razões pelas quais isso pode ser assim. Primeiro, as pessoas podem preferir reduzir sua carga de cálculo; escolhendo certos resultados, elas não têm que calcular probabilidades. Mas o efeito persiste mesmo quando os cálculos envolvidos são muito fáceis. Em segundo lugar, as pessoas podem desconfiar da

legitimidade das probabilidades declaradas. Confio que o lançamento de uma moeda é mais ou menos 50/50 se eu tiver controle sobre a moeda e sobre o lance, mas posso desconfiar do resultado se o lançamento for feito por alguém com interesse no resultado.[6] Na presença de desconfiança, talvez seja melhor ir para a certeza.[7] Terceiro, as pessoas podem estar considerando seu estado emocional, bem como o financeiro. Elas sabem que experimentariam **pesar** se desistissem de certa recompensa (B) por 80% de chance de uma recompensa maior e depois perdessem. Em outras palavras, se A for escolhido, há uma chance de 20% de não obter dinheiro e *sentir-se como um completo idiota*, que é pior do que simplesmente ficar sem dinheiro. Assim, talvez as pessoas que escolhem B sobre A e C sobre D não sejam tão irracionais; estão apenas dizendo que estão dispostas a desistir de $200 de VME para evitar 20% de chance de se sentirem idiotas.

Um problema relacionado é o paradoxo de Ellsberg. Aqui os prêmios são fixos, mas as probabilidades são irrestritas. Sua recompensa vai depender da cor de uma bola escolhida em uma urna. Você sabe que a urna contém 1/3 de bolas vermelhas e 2/3 de bolas pretas ou amarelas, mas você não sabe quantas são pretas e quantas são amarelas. Novamente, perguntam se você prefere a loteria A ou B e, em seguida, C ou D:

A: $100 para uma bola vermelha
B: $100 para uma bola preta
C: $100 para uma bola vermelha ou amarela
D: $100 para uma bola preta ou amarela.

Deve ficar claro que, se acha que há mais bolas vermelhas do que pretas, você deve preferir A sobre B e C sobre D; se acha que há menos vermelhas do que pretas, deve preferir o oposto. Mas acontece que a maioria das pessoas prefere A sobre B e também D sobre C, mesmo que não haja estado do mundo para o qual isso seja racional. Parece que as pessoas têm **aversão à ambiguidade**: A oferece uma chance de ganho de 1/3, enquanto B pode estar em qualquer lugar entre 0 e 2/3. Da mesma forma, D oferece 2/3 de chance, enquanto C pode estar entre 1/3 e 3/3. A maioria das pessoas elege a probabilidade conhecida, em vez de as incógnitas desconhecidas.

Outro problema é que a formulação exata de um problema de decisão pode ter grande impacto sobre as escolhas do agente, o que é chamado de efeito *framing*. A experiência mostra que as pessoas preferem um procedimento médico, que é descrito como tendo "taxa de sobrevivência de 90%" cerca de duas vezes mais do que aquele descrito como tendo "taxa de mortalidade de 10%", mesmo que essas duas declarações signifiquem exatamente a mesma coisa. Essa discrepância no julgamento foi encontrada em experimentos múltiplos e é quase a mesma se os sujeitos forem pacientes de uma clínica, estudantes de uma escola de negócios muito sofisticada ou médicos experientes.

As pessoas sentem-se mais confortáveis fazendo julgamentos de utilidade *relativa* do que absoluta. Eu posso ter pouca ideia de quanto poderia gostar dos diversos vinhos oferecidos por um restaurante.

O restaurante aproveita isso oferecendo uma garrafa de $200 que sabe que ninguém vai comprar, mas que serve para elevar a estimativa de valor de todos os vinhos do cliente e fazer a garrafa de $55 parecer uma pechincha. Isso se chama **efeito de ancoragem**.

Se informantes humanos insistem em juízos de preferência contraditória, não há nada que os agentes automatizados possam fazer para ser coerentes com eles. Felizmente, os julgamentos de preferência feitos por seres humanos muitas vezes são passíveis de revisão, à luz de uma análise mais aprofundada. Paradoxos, como o paradoxo de Allais, são muito reduzidos (mas não eliminados) se as escolhas forem mais bem explicadas. No trabalho na Harvard Business School sobre a avaliação da utilidade do dinheiro, Keeney e Raiffa (1976, p. 210) encontraram o seguinte:

> Os indivíduos tendem a ser muito avessos ao risco nos pequenos e, portanto (…) as funções utilidade adequadas exibem prêmios de risco muito grande e inaceitável para loterias com grande envergadura… A maioria dos indivíduos, no entanto, pode conciliar as inconsistências e sentir que aprendeu uma lição importante sobre como querem se comportar. Como consequência, alguns indivíduos cancelam seu seguro automobilístico contra colisão e acrescentam mais cláusulas em seu seguro de vida.

A evidência para a irracionalidade humana também é questionada por pesquisadores no campo da **psicologia evolucionária**, que aponta para o fato de que os mecanismos de tomada de decisão do nosso cérebro não evoluíram para resolver problemas de palavras com probabilidades e prêmios declarados como números decimais. Admitamos, por causa do argumento, que o cérebro tem um mecanismo neural embutido para calcular probabilidades e utilidades ou algo funcionalmente equivalente; em caso afirmativo, os insumos necessários seriam obtidos através da experiência acumulada de resultados e recompensas, em vez de através de apresentações linguísticas de valores numéricos. Está longe de ser óbvio que podemos acessar diretamente os mecanismos neurais embutidos no cérebro apresentando problemas de decisão na forma linguística/numérica. O próprio fato de que formulações diferentes do *mesmo problema de decisão* provocam escolhas diferentes sugere que o problema de decisão em si não conseguiu aprovação. Estimulados por essa observação, os psicólogos têm tentado apresentar problemas de raciocínio incerto e tomada de decisão em forma de "evolução adequada"; por exemplo, em vez de dizer " taxa de sobrevivência de 90%", o experimentador pode mostrar 100 figuras de animação estilizadas, onde o paciente morre em 10 delas e sobrevive em 90. (O tédio é um fator complicador nesses experimentos!) Com problemas de decisão colocados dessa maneira, as pessoas parecem ficar muito mais perto de um comportamento racional do que anteriormente se suspeitava.

16.4 FUNÇÕES UTILIDADE MULTIATRIBUTO

A tomada de decisões no campo de política pública envolve altos riscos, tanto em dinheiro como em vidas. Por exemplo, ao decidir quais níveis de emissões nocivas permitir de uma fábrica poderosa, os legisladores devem ponderar a prevenção de morte e incapacidade em relação ao benefício do poder e da carga econômica de mitigar as emissões. O projeto de um novo aeroporto exige que seja considerada a ruptura causada pela construção, o custo do terreno, a distância até os centros populacionais, o ruído das operações de voo, questões de segurança relacionadas à topografia local e às condições meteorológicas, e assim por diante. Problemas como esses, em que os resultados são caracterizados por dois ou mais atributos, são manipulados pela **teoria de**

utilidade multiatributo.

Chamaremos os atributos de $\mathbf{X} = X_1, \ldots, X_n$; um vetor completo de atribuições será $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$. onde cada x_i é um valor numérico ou um valor discreto com ordenação de valores assumida. Vamos supor que os valores mais altos de um atributo correspondam às utilidades mais altas, todas as outras coisas sendo iguais. Por exemplo, se escolhermos *AusênciaDeRuído* como um atributo no problema de aeroporto, quanto maior seu valor, melhor a solução.[8] Começamos examinando casos em que as decisões podem ser tomadas *sem* combinar os valores de atributo em um único valor de utilidade. Em seguida, examinaremos casos em que as utilidades de combinações de atributos podem ser especificadas de forma muito concisa.

16.4.1 Dominância

Vamos supor que o local do aeroporto S_1 custe menos, gere menos poluição sonora e seja mais seguro que o local S_2. Ninguém hesitaria em rejeitar S_2. Então, dizemos que existe uma **dominância estrita** de S_1 sobre S_2. Em geral, se uma opção tiver valor mais baixo que alguma outra opção em todos os atributos, ela não precisará de consideração adicional. Com frequência, a dominância estrita é muito útil no estreitamento do campo de opções para os competidores reais, embora raramente resulte em uma única escolha. A Figura 16.4(a) mostra um diagrama esquemático para o caso de dois atributos.

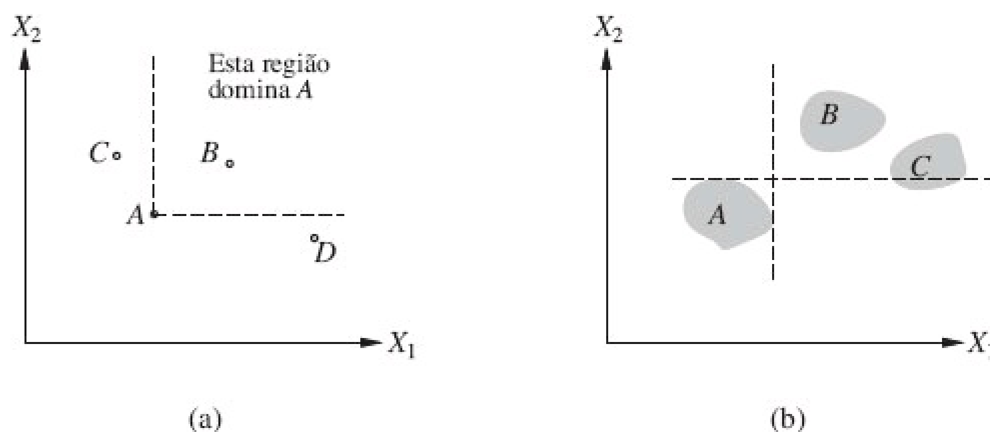


Figura 16.4 Dominância estrita. (a) Determinística: a opçãoA é estritamente dominada por B, mas não por C ou D. (b) Incerta: A é estritamente dominado por B, mas não por C.

Isso é ótimo para o caso determinístico, em que os valores de atributos são conhecidos com certeza. E o caso geral, em que os resultados das ações são incertos? Pode ser construída uma analogia direta da dominância estrita em que, apesar da incerteza, todos os resultados concretos possíveis para S_1 dominam estritamente todos os resultados possíveis para S_2 (veja a Figura 16.4(b)). É claro que isso provavelmente ocorrerá com frequência ainda menor que no caso determinístico.

Felizmente, existe uma generalização mais útil chamada **dominância estocástica**, que ocorre com muita frequência em problemas reais. É mais fácil compreender a dominância estocástica no contexto de um único atributo. Vamos supor que acreditamos que o custo da localização do aeroporto em S_1

esteja uniformemente distribuído entre \$2,8 bilhões e \$4,8 bilhões, e que o custo em S_2 esteja uniformemente distribuído entre \$3 bilhões e \$5,2 bilhões. A Figura 16.5(a) mostra essas distribuições, com o custo representado como um valor negativo. Então, dada apenas a informação de que a utilidade diminui com o custo, podemos afirmar que S_1 domina estocasticamente S_2 (isto é, S_2 pode ser descartado). É importante observar que isso *não* decorre da comparação entre os custos esperados. Por exemplo, se soubéssemos que o custo de S_1 é *exatamente* \$3,8 bilhões, seríamos *incapazes* de tomar uma decisão sem informações adicionais sobre a utilidade do dinheiro.

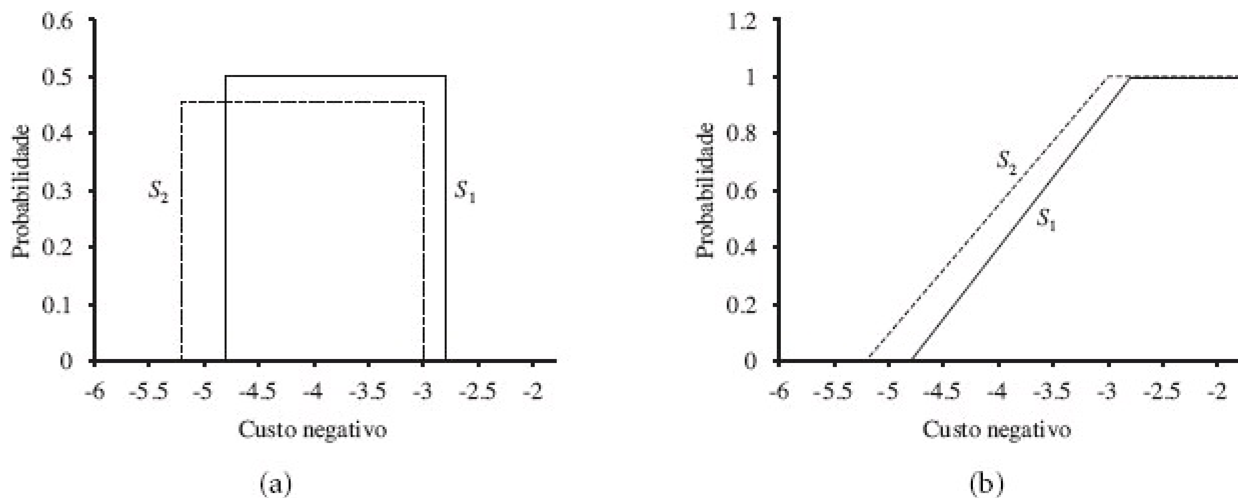


Figura 16.5 Dominância estocástica. (a) S_1 domina estocasticamente S_2 em custo. (b) Distribuições cumulativas para o custo negativo de S_1 e S_2.

(Pode parecer estranho que *mais* informações sobre o custo de S_1 poderia fazer o agente *menos* capaz de decidir. O paradoxo é resolvido observando que, na ausência de informações exatas de custo, a decisão é mais fácil de tomar, mas é mais provável que esteja errada.)

O relacionamento exato entre as distribuições de atributos necessárias para estabelecer a dominância estocástica é mais bem visualizada examinando-se as **distribuições cumulativas**, mostradas na Figura 16.5(b) (veja também o Apêndice A). A distribuição cumulativa mede a probabilidade de que o custo seja menor ou igual a qualquer valor dado, ou seja, ela integra a distribuição original. Se a distribuição cumulativa para S_1 estiver sempre à direita da distribuição cumulativa para S_2, em termos estocásticos S_1 será mais econômico que S_2. Formalmente, se duas ações A_1 e A_2 resultam em distribuições de probabilidade $p_1(x)$ e $p_2(x)$ sobre o atributo X, então A_1 dominará estocasticamente A_2 sobre X se:

$$\forall x \quad \int_{-\infty}^{x} p_1(x')\, dx' \leq \int_{-\infty}^{x} p_2(x')\, dx' \,.$$

☞ A relevância dessa definição para a seleção de decisões ótimas vem da seguinte propriedade: *se A_1 domina estocasticamente A_2, então, para qualquer função utilidade monotonicamente não decrescente $U(x)$, a utilidade esperada de A_1 é pelo menos tão alta quanto a utilidade esperada de*

A_2. Em consequência disso, se uma ação é estocasticamente dominada por outra ação em todos os atributos, ela pode ser descartada.

A condição de dominância estocástica pode parecer bastante técnica e talvez não muito fácil de avaliar sem cálculos extensivos de probabilidade. De fato, ela pode ser decidida com muita facilidade em vários casos. Por exemplo, suponha que o custo de transporte da construção dependa da distância até o fornecedor. O custo em si é incerto, mas, quanto maior a distância, maior o custo. Se S_1 estiver mais próximo que S_2, S_1 dominará S_2 em custo. Existem algoritmos — embora eles não sejam apresentados aqui — que realizam a propagação dessa espécie de informação qualitativa entre variáveis incertas em **redes probabilísticas qualitativas**, permitindo a um sistema tomar decisões racionais baseadas em dominância estocástica, sem utilizar valores numéricos.

16.4.2 Estrutura de preferências e utilidade multiatributo

Vamos supor que tenhamos n atributos, cada um dos quais com d valores distintos possíveis. Para especificar a função utilidade completa $U(x_1,\dots, x_n)$, precisamos de d^n valores no pior caso. Agora, o pior caso corresponde a uma situação em que as preferências do agente não têm absolutamente nenhuma regularidade. A teoria da utilidade multiatributo se baseia na suposição de que as preferências de agentes típicos têm muito mais estrutura que isso. A abordagem básica é identificar regularidades no comportamento de preferências que esperaríamos ver e utilizar o que chamamos **teoremas de representação** para mostrar que um agente com certo tipo de estrutura de preferências tem uma função utilidade

$$U(x_1,\dots,x_n) = F[f_1(x_1),\dots, f_n(x_n)],$$

onde F é, esperamos, uma função simples como a adição. Note a semelhança com o uso de redes bayesianas para decompor a probabilidade conjunta de diversas variáveis aleatórias.

Preferências sem incerteza

Vamos começar com o caso determinístico. Lembre-se de que, para ambientes determinísticos, o agente tem uma função de valor $V(x_1,\dots, x_n)$; o objetivo é representar essa função de forma concisa. Aregularidade básica que surge em estruturas de preferências determinísticas é chamada **independência de preferências**. Dois atributos X_1 e X_2 são preferencialmente independentes de um terceiro atributo X_3 se a preferência entre resultados $\langle x_1, x_2, x_3\rangle$ e $\langle x_1', x_2', x_3\rangle$ não depende do valor específico x_3 para o atributo X_3.

Voltando ao exemplo do aeroporto, onde temos (entre outros atributos) *Ruído*, *Custo* e *Mortes* a considerar, alguém poderia propor que *Ruído* e *Custo* sejam preferencialmente independentes de *Mortes*. Por exemplo, se preferirmos um estado com 20.000 pessoas residentes na rota de voos e um custo de construção de $4 bilhões a um estado com 70.000 pessoas residentes na rota de voos e um custo de $3,7 bilhões quando o nível de segurança é 0,06 morte por milhão de milhas de passageiros em ambos os casos, teremos a mesma preferência quando o nível de segurança for 0,12 ou 0,03, e a mesma independência seria válida para preferências entre qualquer outro par de valores de *Ruído* e

Custo. Também é aparente que *Custo* e *Mortes* são preferencialmente independentes de *Ruído*, e que *Ruído* e *Mortes* são preferencialmente independentes de *Custo*. Dizemos que o conjunto de atributos {*Ruído*, *Custo*, *Mortes*} exibe **independência preferencial mútua** (IPM). A IPM afirma que, embora cada atributo possa ser importante, não afeta os compromissos que os outros atributos mantêm entre si.

A independência preferencial mútua é uma expressão que parece complicada, mas, graças a um importante teorema devido ao economista Gérard Debreu (1960), podemos derivar a partir dela uma forma muito simples para a função de valor do agente: se os atributos $X_1, \ldots, X_n$ *guardam entre si uma independência preferencial mútua, então o comportamento de preferências do agente pode ser descrito como a maximização da função*:

$$V(x_1, \ldots, x_n) = \sum_i V_i(x_i),$$

onde cada Vi é uma função de valor que se refere apenas ao atributo X_i. Por exemplo, talvez a decisão sobre o local do aeroporto pudesse ser tomada com o uso de uma função de valor:

$$V(ruído, custo, mortes) = -ruído \times 10^4 - custo - mortes \times 10^{12}.$$

Uma função de valor desse tipo é chamada **função de valor aditiva**. As funções aditivas constituem um modo extremamente natural de se descrever a função de valor de um agente, e são válidas em muitas situações reais. Para *n* atributos, avaliar uma função de valor aditivo exige avaliar em separado funções de valor *n* unidimensional, em vez de uma função *n* dimensional; normalmente, isso representa uma redução exponencial do número de experimentos de preferência que são necessários. Mesmo quando a IPM não é estritamente válida, como poderia ocorrer no caso de valores extremos dos atributos, uma função de valor aditiva ainda poderia fornecer boa aproximação para as preferências do agente. Isso é especialmente verdadeiro quando as violações da IPM ocorrem em porções dos intervalos de atributos que têm pouca probabilidade de ocorrerem na prática.

Para entender melhor o IPM, ajuda examinar os casos em que *não são* válidos. Suponha que você esteja em um mercado medieval, considerando a compra de alguns cães de caça, algumas galinhas e algumas gaiolas de vime para as galinhas. Os cães de caça são muito valiosos, mas, se você não tiver gaiolas suficientes para as galinhas, os cachorros vão comer as galinhas; daí, a troca entre cães e galinhas depende fortemente do número de gaiolas, e o IPM será violado. A existência desses tipos de interações entre vários atributos torna muito mais difícil avaliar a função de valor global.

Preferências com incerteza

Quando a incerteza estiver presente no domínio, também precisaremos considerar a estrutura de preferências entre loterias e entender as propriedades resultantes de funções utilidade, e não apenas de funções de valor. A matemática desse problema pode se tornar bastante complicada e, assim, apresentaremos apenas um dos principais resultados para dar uma ideia do que pode ser feito. O leitor deve consultar o trabalho de Keeney e Raiffa (1976) para ver um estudo completo do campo.

A noção básica de **independência da utilidade** estende a independência de preferências para

cobrir as loterias: um conjunto de atributos **X** é independente da utilidade de um conjunto de atributos **Y** se as preferências entre loterias sobre os atributos em **X** são independentes dos valores específicos dos atributos em **Y**. Um conjunto de atributos é **mutuamente independente da utilidade** (MIU) se cada um de seus subconjuntos é independente da utilidade dos atributos restantes. Mais uma vez, parece razoável propor que os atributos do aeroporto sejam MIU.

A MIU implica que o comportamento do agente pode ser descrito com o uso de uma **função utilidade multiplicativa** (Keeney, 1974). A forma geral de uma função utilidade multiplicativa é mais bem visualizada observando-se o caso correspondente a três atributos. Por concisão, utilizaremos U_i para representar $U_i(x_i)$:

$$U = k_1U_1 + k_2U_2 + k_3U_3 + k_1k_2U_1U_2 + k_2k_3U_2U_3 + k_3k_1U_3U_1 + k_1k_2k_3U_1U_2U_3.$$

Embora não pareça muito simples, essa expressão contém apenas três funções utilidade de um único atributo e três constantes. Em geral, um problema de *n* atributos que exibe mil pode ser modelado com a utilização de *n* utilidades de um único atributo e *n* constantes. Cada uma das funções utilidade de um único atributo pode ser desenvolvida independentemente dos outros atributos, e essa combinação oferecerá a garantia de gerar as preferências globais corretas. São necessárias suposições adicionais para se obter uma função utilidade puramente aditiva.

16.5 REDES DE DECISÃO

Nesta seção, examinaremos um mecanismo geral para a tomada de decisões racionais. Com frequência, a notação é chamada **diagrama de influência** (Howard e Matheson, 1984), mas utilizaremos a expressão mais descritiva **rede de decisão**. As redes de decisão combinam redes bayesianas com tipos de nós adicionais para ações e utilidades. Usaremos a localização do aeroporto como exemplo.

16.5.1 Representação de um problema de decisão com uma rede de decisão

Em sua forma mais geral, uma rede de decisão representa informações sobre o estado atual do agente, suas ações possíveis, o estado que resultará da ação do agente e a utilidade desse estado. Portanto, ela fornece um substrato para a implementação de agentes baseados em utilidade do tipo apresentado primeiro na Seção 2.4.5. A Figura 16.6 mostra uma rede de decisão para o problema de localização do aeroporto. Ela ilustra os três tipos de nós utilizados:

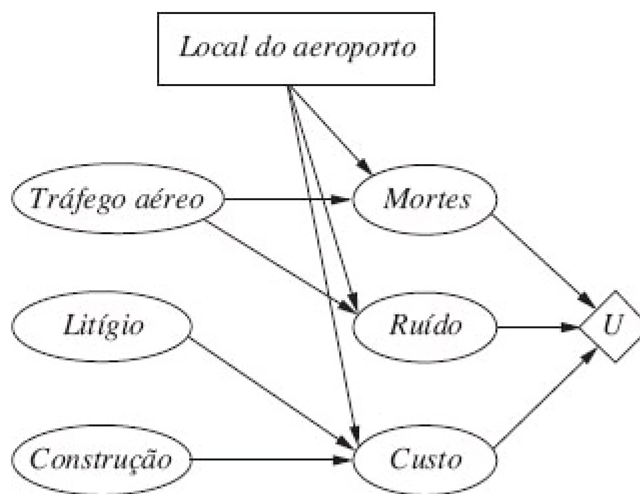


Figura 16.6 Uma rede de decisão simples para o problema de localização do aeroporto.

- **Nós de acaso** (elipses) representam variáveis aleatórias, da mesma maneira que nas redes bayesianas. O agente poderia estar inseguro sobre o custo da construção, o nível de tráfego aéreo e o potencial para litígio, e as variáveis *Mortes*, *Ruído* e *Custo* total, cada uma das quais também depende do local escolhido. Cada nó de acaso está associado a uma distribuição condicional que é indexada pelo estado dos nós pais. Em redes de decisão, os nós pais podem incluir nós de decisão, bem como nós de acaso. Observe que cada um dos nós de acaso do estado atual poderia fazer parte de uma grande rede bayesiana para avaliar os custos de construção, os níveis de tráfego aéreo ou os potenciais de litígio.
- **Nós de decisão** (retângulos) representam pontos em que o tomador de decisão tem a possibilidade de escolher ações. Nesse caso, a ação *LocalAeroporto* pode assumir um valor diferente para cada local que está sendo considerado. A escolha influencia o custo, a segurança e o ruído resultantes. Neste capítulo, vamos supor que estamos lidando com um único nó de decisão. O Capítulo 17 lida com casos em que deve ser tomada mais de uma decisão.
- **Nós de utilidade** (losangos) representam a função utilidade do agente.[9] O nó de utilidade tem como pais todas as variáveis que descrevem o resultado que afeta diretamente a utilidade. Associada ao nó de utilidade, encontramos uma descrição da utilidade do agente como uma função dos atributos do pai. A descrição poderia ser simplesmente uma tabulação da função ou uma função parametrizada aditiva ou multilinear.

Em muitos casos, também é utilizada uma forma simplificada. A notação permanece idêntica, mas os nós de acaso que descrevem o estado resultante são omitidos. Em vez disso, o nó de utilidade é conectado diretamente aos nós do estado atual e ao nó de decisão. Nesse caso, em vez de representar uma função utilidade sobre estados, o nó de utilidade representa a utilidade *esperada* associada a cada ação, conforme definimos na Equação 16.1, ou seja, o nó está associado com uma **função ação-utilidade** (também conhecida como **Q-função** em aprendizado por reforço, como descrito no Capítulo 21). A Figura 16.7 mostra a representação de ação-utilidade do problema de aeroporto.

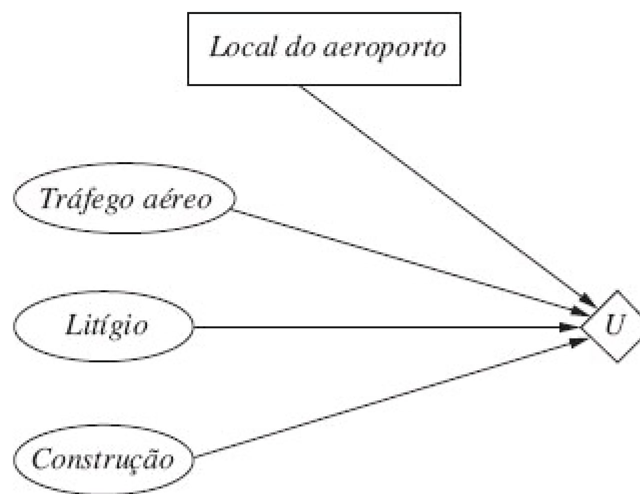


Figura 16.7 Representação simplificada do problema de localização do aeroporto. Os nós de acaso que correspondem a estados resultantes foram fatorados.

Note que, pelo fato de os nós de acaso *Ruído*, *Mortes* e *Custo* da Figura 16.6 se referirem a estados futuros, eles nunca podem ter seus valores definidos como variáveis de evidência. Desse modo, a versão simplificada que omite esses nós pode ser empregada sempre que a forma mais geral puder ser utilizada. Embora a forma simplificada contenha menos nós, a omissão de uma descrição explícita do resultado da decisão sobre a localização significa que ela é menos flexível com relação a mudanças nas circunstâncias. Por exemplo, na Figura 16.6, uma mudança nos níveis de ruído das aeronaves pode se refletir em uma alteração na tabela de probabilidade condicional associada ao nó *Ruído*, enquanto uma mudança no peso acordado para a poluição sonora na função utilidade pode se refletir em uma mudança na tabela de utilidade. Por outro lado, no diagrama de ação-utilidade, apresentado na Figura 16.7, todas essas mudanças têm de ser refletidas por alterações na tabela de ação-utilidade. Em essência, a formulação de ação-utilidade é uma versão *compilada* da formulação original.

16.5.2 Avaliação de redes de decisão

As ações são selecionadas pela avaliação da rede de decisão correspondente a cada configuração possível do nó de decisão. Uma vez que o nó de decisão é estabelecido, ele se comporta exatamente como um nó de acaso que tenha sido definido como uma variável de evidência. O algoritmo para avaliar redes de decisão é dado a seguir:

1. Definir as variáveis de evidência para o estado atual.
2. Para cada valor possível do nó de decisão:
   (a) Definir o nó de decisão com esse valor.
   (b) Calcular as probabilidades posteriores para os nós pais do nó de utilidade, usando um algoritmo-padrão de inferência probabilística.
   (c) Calcular a utilidade resultante para a ação.
3. Retornar a ação com a utilidade mais alta.

Essa é uma extensão direta do algoritmo de rede bayesiana e pode ser diretamente incorporada ao

projeto de agente apresentado na Figura 13.1. Veremos no Capítulo 17 que a possibilidade de executar diversas ações em sequência torna o problema muito mais interessante.

16.6 O VALOR DA INFORMAÇÃO

Na análise precedente, partimos do princípio de que todas as informações relevantes, ou pelo menos todas as informações disponíveis, são fornecidas ao agente antes de ele tomar sua decisão. Na prática, isso dificilmente acontece. *Uma das partes mais importantes da tomada de decisões é saber que perguntas formular*. Por exemplo, um médico não pode esperar ter o resultado de *todos os possíveis* exames e questões de diagnóstico no momento em que um paciente entra pela primeira vez no consultório.[10] Muitas vezes, os exames são dispendiosos, e às vezes arriscados (tanto diretamente quanto devido a retardos associados). Sua importância depende de duas variáveis: do fato de os resultados dos exames levarem ou não a um plano de tratamento significativamente melhor e da probabilidade de cada um dos diversos resultados para os exames.

Esta seção descreve a **teoria do valor da informação**, que permite a um agente escolher que informação adquirir. Assumimos que, antes de escolher uma ação "real" representada pelo nó de decisão, o agente pode adquirir o valor de qualquer uma das variáveis de acaso potencialmente observáveis no modelo. Assim, a teoria do valor da informação envolve uma forma simplificada de tomada decisão sequencial — simplificada, pois as ações de observação afetam apenas **o estado de crença** do agente, não o estado físico externo. O valor de qualquer observação particular deve derivar do potencial de afetar a eventual ação física do agente; e esse potencial pode ser estimado diretamente a partir do modelo de decisão em si.

16.6.1 Um exemplo simples

Vamos supor que uma empresa petrolífera espere comprar um dos n blocos indistinguíveis de direitos de perfuração oceânica. Vamos supor ainda que exatamente um dos blocos contenha petróleo no valor de C dólares, enquanto os outros não têm valor. O preço inicial de cada bloco é C/n dólares. Se a empresa for de risco neutro, ela será indiferente entre comprar e não comprar um bloco.

Agora, suponha que um sismólogo ofereça à empresa os resultados de uma sondagem do bloco número 3, que indica definitivamente se o bloco contém petróleo. Quanto a empresa deve estar disposta a pagar pela informação? O caminho para responder a essa pergunta é examinar o que a empresa faria se tivesse a informação:

- Com probabilidade $1/n$, a sondagem indicará petróleo no bloco 3. Nesse caso, a empresa comprará o bloco 3 por C/n dólares e terá um lucro de $C - C/n = (n - 1)C/n$ dólares.
- Com probabilidade $(n - 1)/n$, a sondagem mostrará que o bloco não contém petróleo e, nesse caso, a empresa comprará um bloco diferente. Agora a probabilidade de encontrar petróleo em um dos outros blocos muda de $1/n$ para $1/(n - 1)$ e, assim, a empresa obtém um lucro esperado de $C/(n - 1) - C/n = C/n(n - 1)$ dólares.

Agora podemos calcular o lucro esperado, dadas as informações da sondagem:

$$\frac{1}{n} \times \frac{(n-1)C}{n} + \frac{n-1}{n} \times \frac{C}{n(n-1)} = C/n\,.$$

Então, a empresa deve estar disposta a pagar ao sismólogo até *C*/*n* dólares pela informação: a informação é tão valiosa quanto o próprio bloco.

O valor da informação deriva do fato de que, *com* a informação, um curso de ação pode ser alterado para se adaptar à situação *real*. É possível decidir de acordo com a situação; por outro lado, sem a informação, a decisão tem de ser tomada avaliando-se a melhor opção em média sobre as situações possíveis. Em geral, o valor de dado item de informação é definido como a diferença de valor esperado entre as ações antes e depois da obtenção da informação.

16.6.2 Fórmula geral da informação perfeita

É simples derivar uma fórmula matemática geral para o valor da informação. Em geral, supomos que a evidência exata seja obtida sobre o valor de alguma variável aleatória E_j e, assim, é usada a expressão **valor da informação perfeita** (VIP).[11]

Seja **e** o conhecimento atual do agente. Então, o valor da melhor ação atual α é definido por:

$$UE(\alpha|\mathbf{e}) = \max_a \sum_{s'} P(\text{RESULTADO}\,(a) = s' \mid a, \mathbf{e})\, U(s')\,,$$

e o valor da melhor ação nova (após a nova evidência $E_j = e_j$ ser obtida) será:

$$UE(\alpha_{e_j}|\mathbf{e}, e_j) = \max_a \sum_{s'} P(\text{RESULTADO}(a) = s' \mid a, \mathbf{e}, e_j)\, U(s')\,.$$

Porém, E_j é uma variável aleatória cujo valor é *atualmente* desconhecido; assim, para determinar o valor de E_j, dada uma informação atual **e** devemos calcular a média sobre todos os valores possíveis e_{jk} que poderíamos descobrir para E_j utilizando nossas crenças *atuais* sobre seu valor:

$$VPI_{\mathbf{e}}(E_j) = \left(\sum_k P(E_j = e_{jk}|\mathbf{e})\; UE(\alpha_{e_{jk}}|\mathbf{e}, E_j = e_{jk})\right) - UE(\alpha|\mathbf{e})\,.$$

Para obter alguma intuição relativa a essa fórmula, considere o caso simples em que existem apenas duas ações, α_1 e α_2, da qual devemos escolher uma. Suas utilidades esperadas atuais são U_1 e U_2. As informações $E_j = e_{jk}$ produzirão algumas novas utilidades esperadas U_1'' e U_2'' para as ações; porém, antes de obter E_j, teremos algumas distribuições de probabilidade sobre os valores possíveis de U_1'' e U_2'' (que vamos supor independentes).

Suponha que a_1 e a_2 representem duas rotas diferentes através de uma cordilheira no inverno. a_1 é

uma ótima estrada reta que passa por um vale, e a_2 é uma estrada poeirenta e sinuosa, próxima ao cume das montanhas.

Apenas com essas informações, a_1 é claramente preferível porque é bastante provável que a_2 esteja bloqueada por avalanches, enquanto é improvável que algo bloqueie a_1. Então, U_1 é sem dúvida mais alta que U_2. É possível obter relatórios E_j sobre o estado real de cada estrada, o que nos dará novas expectativas, U'_1 e U'_2, relativas às duas rotas. As distribuições para essas expectativas são mostradas na Figura 16.8(a). É óbvio que, nesse caso, não vale a pena a despesa de obter relatórios de satélites porque é improvável que as informações que derivam deles alterem o plano. Sem mudança, a informação não tem nenhum valor.

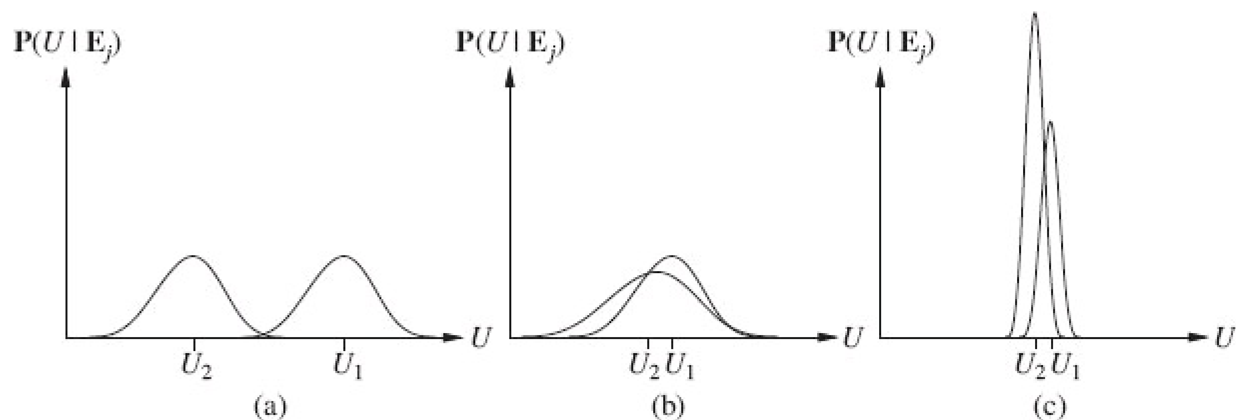


Figura 16.8 Três casos genéricos para o valor da informação. Em (a), a_1 quase certamente permanecerá superior a a_2 e, assim, a informação não é necessária. Em (b), a escolha é obscura e a informação é crucial. Em (c), a escolha é obscura, mas, como faz pouca diferença, a informação é menos valiosa. (Observação: o fato de U_2 ter um pico alto em (c) significa que seu valor esperado é conhecido com maior certeza que em U_1.)

Agora vamos supor que estamos escolhendo entre duas estradas sinuosas distintas, de comprimentos ligeiramente diferentes e transportando um passageiro seriamente ferido. Então, mesmo quando U_1 e U_2 estão muito próximos, as distribuições de U'_1 e U'_2 são muito amplas. Existe possibilidade significativa de que a segunda rota acabe ficando limpa enquanto a primeira se mantém bloqueada e, nesse caso, a diferença de utilidade será muito alta. A fórmula de VIP indica que talvez compense obter os relatórios do satélite. Tal situação é mostrada na Figura 16.8(b).

Finalmente, suponha que estejamos escolhendo entre duas estradas poeirentas no verão, quando o bloqueio por avalanches é improvável. Nesse caso, os relatórios dos satélites poderiam mostrar que, em uma das estradas, encontraremos uma paisagem mais bonita que a da outra porque ela passa por prados alpinos floridos ou, então, talvez uma delas seja mais úmida porque tem muitos córregos. Provavelmente mudaríamos nossos planos se tivéssemos essas informações. Entretanto, nesse caso, a diferença de valor entre as duas rotas talvez ainda fosse muito pequena, de forma que não nos preocuparemos em obter os relatórios. Essa situação é mostrada na Figura 16.8(c).

Em resumo, *a informação tem valor até o ponto em que apresenta alguma probabilidade de causar uma mudança de planos e até o ponto em que o novo plano é significativamente melhor que o velho*.

16.6.3 Propriedades do valor da informação

Poderíamos perguntar se é possível a informação ser prejudicial: ela pode realmente ter valor esperado negativo? Intuitivamente, devemos esperar que isso seja impossível. Afinal, no pior caso, poderíamos simplesmente ignorar a informação e fingir que nunca a recebemos. Isso é confirmado pelo teorema a seguir, que se aplica a qualquer agente de teoria da decisão:

☞ *O valor da informação é não negativo*:

$$\forall \mathbf{e}, E_j \quad VPI_{\mathbf{e}}(E_j) \geq 0 \,.$$

O teorema decorre diretamente da definição de VIP, e deixamos a prova como exercício (Exercício16.18). Naturalmente, é um teorema sobre o valor *esperado*, não sobre o valor *real*. Informação adicional pode conduzir facilmente a um plano que *acaba* por ser pior do que o plano original, se acontecer de a informação ser enganosa. Por exemplo, um exame médico que dá resultado falso-positivo pode levar a cirurgias desnecessárias; mas isso não significa que o teste não deva ser feito.

É importante lembrar que o VIP depende do estado atual da informação, e é esse o motivo pelo qual ele tem um subscrito. Ele pode mudar à medida que mais informações são adquiridas. Para qualquer evidência determinada de E_j, o valor de aquisição pode cair (por exemplo, se outra variável restringe fortemente a posterior por E_j) ou subir (por exemplo, se outra variável fornece uma pista em que E_j se desenvolva, possibilitando que um plano novo e melhor seja concebido).

Desse modo, VIP é não aditivo, isto é:

$$VPI_{\mathbf{e}}(E_j, E_k) \neq VPI_{\mathbf{e}}(E_j) + VPI_{\mathbf{e}}(E_k) \quad \text{(em geral)} \,.$$

Entretanto, o VIP é independente da ordem. Ou seja:

$$VPI_{\mathbf{e}}(E_j, E_k) = VPI_{\mathbf{e}}(E_j) + VPI_{\mathbf{e},e_j}(E_k) = VPI_{\mathbf{e}}(E_k) + VPI_{\mathbf{e},e_k}(E_j) \,.$$

A independência da ordem distingue ações de detecção de ações comuns e simplifica o problema de calcular o valor de uma sequência de ações de detecção.

16.6.4 Implementação de um agente de coleta de informações

Um agente sensato deve formular as perguntas do usuário em uma ordem razoável, deve evitar formular perguntas irrelevantes, deve levar em conta a importância de cada fragmento de informação em relação a seu custo e deve parar de formular perguntas quando isso for apropriado. Todos esses recursos podem ser alcançados com o uso do valor da informação como guia.

A Figura 16.9 mostra o projeto global de um agente que pode coletar informações de forma inteligente antes de agir. Por enquanto, vamos supor que, a cada variável de evidência observável E_j,

existe um custo associado, $Custo(E_j)$, que reflete o custo da obtenção da evidência através de testes, consultores, perguntas ou qualquer outro processo. O agente solicita o que parece ser o item de informação mais valioso, comparado a seu custo. Supomos que, como resultado da ação $Solicitar(E_j)$, a percepção seguinte forneça o valor de E_j. Se nenhuma observação compensar seu custo, o agente selecionará uma ação "real".

função AGENTE-DE-COLETA-DE-INFORMAÇÕES(*percepção*) **retorna** uma *ação*
  variáveis estáticas: *D*, uma rede de decisão

  integrar *percepção* a *D*
  $j \leftarrow$ o valor que maximiza $VIP(E_j) / Custo(E_j)$
  se $VIP(E_j) > Custo(E_j)$
    então retornar SOLICITAR(E_j)
  senão retornar a melhor ação a partir de *D*

Figura 16.9 Projeto de um agente de coleta de informações. O agente funciona selecionando repetidamente a observação com o mais alto valor de informação, até o custo da próxima observação ser maior que o seu benefício esperado.

O algoritmo de agente que descrevemos implementa uma forma de coleta de informações chamada **míope**. Ela recebe esse nome porque utiliza a fórmula do VIP de maneira limitada, calculando o valor da informação como se apenas uma única variável de evidência fosse adquirida. O controle míope se baseia na mesma ideia de heurística da pesquisa gulosa, e com frequência funciona bem na prática (por exemplo, mostrou- se que ele supera os médicos especialistas na seleção de exames de diagnóstico).

Porém, se não houver nenhuma única variável de evidência que ajude muito, um agente míope pode apressadamente tomar uma ação quando teria sido melhor pedir duas ou mais variáveis primeiro e depois agir. Uma abordagem melhor nessa situação seria a construção de um *plano condicional* (conforme descrito na Seção 11.3.2) que pede os valores das variáveis e escolhe os próximos passos, dependendo da resposta.

Uma consideração final é o efeito que uma série de questões terá sobre um entrevistado humano. As pessoas podem responder melhor a uma série de perguntas, se elas "fizerem sentido", de modo que alguns sistemas especialistas são construídos para levar isso em conta, fazendo perguntas em uma ordem que maximiza a utilidade total do sistema e dos humanos em vez de uma ordem que maximiza o valor da informação.

16.7 SISTEMAS ESPECIALISTAS DE TEORIA DA DECISÃO

O campo da **análise da decisão**, que evoluiu nas décadas de 1950 e 1960, estuda a aplicação da teoria da decisão a problemas reais de decisão. Ela é utilizada para ajudar na tomada de decisões racionais em domínios importantes em que os riscos são altos, como os de negócios, governo, leis,

estratégia militar, diagnóstico médico e saúde pública, projetos de engenharia e gerenciamento de recursos. O processo envolve um estudo cuidadoso das ações e resultados possíveis, bem como as preferências estabelecidas para cada resultado. É tradicional na análise da decisão mencionar dois papéis: o **tomador de decisões** enuncia preferências entre resultados e o **analista de decisões** enumera as ações e os resultados possíveis, e se baseia nas preferências do tomador de decisões para determinar o melhor curso de ação. Até o início da década de 1980, o principal objetivo da análise da decisão era ajudar os seres humanos a tomarem decisões que realmente refletissem suas próprias preferências. Como cada vez mais processos de decisão tornam-se automatizados, a análise de decisão é cada vez mais utilizada para garantir que os processos automatizados se comportarão como desejado.

As primeiras pesquisas de sistemas especialistas se concentravam em responder a perguntas, e não na tomada de decisões. Esses sistemas, que recomendavam ações em vez de fornecer opiniões sobre as questões em geral, faziam isso utilizando regras de condição-ação, em vez de empregarem representações explícitas de resultados e preferências. O surgimento das redes bayesianas, no final da década de 1980, tornou possível a construção de sistemas em grande escala que geravam inferências probabilísticas consistentes a partir da evidência. A adição de redes de decisão significa que podem ser desenvolvidos sistemas especialistas que recomendem decisões ótimas, refletindo as preferências do usuário, bem como a evidência disponível.

Um sistema que incorpora utilidades pode evitar uma das armadilhas mais comuns associadas ao processo de consulta: confundir probabilidade e importância. Por exemplo, uma estratégia comum nos primeiros sistemas especialistas médicos era classificar os diagnósticos possíveis em ordem de probabilidade e informar o mais provável. Infelizmente, isso pode ser desastroso! Para a maioria dos pacientes em geral, os dois diagnósticos mais *prováveis* são normalmente "não há nada de errado com você" e "você tem um forte resfriado"; porém, se o terceiro diagnóstico mais provável para determinado paciente for câncer de pulmão, esse será um assunto sério. É óbvio que um plano de exames ou de tratamento deve depender tanto de probabilidades quanto de utilidades. Os sistemas especialistas médicos atuais podem levar em conta o valor da informação para recomendar testes e, em seguida, descrever um diagnóstico diferencial.

Descreveremos agora o processo de engenharia do conhecimento para sistemas especialistas de teoria da decisão. Como exemplo, consideraremos o problema de selecionar um tratamento médico para uma espécie de doença congênita do coração em crianças (veja Lucas, 1996).

Cerca de 0,8% das crianças nascem com uma anomalia do coração, sendo a mais comum o **estreitamento da aorta** (uma constrição da aorta). Essa anomalia pode ser tratada com cirurgia, angioplastia (expandindo-se a aorta com um balão inserido na artéria) ou medicação. O problema é decidir que tratamento usar e quando fazê-lo: quanto mais jovem for a criança, maiores serão os riscos de certos tratamentos, mas não se deve esperar muito tempo. Um sistema especialista de teoria da decisão para esse problema pode ser criado por uma equipe que consiste em um especialista em pelo menos um domínio (um cardiologista pediátrico) e um engenheiro do conhecimento. O processo pode ser dividido nas etapas a seguir:

Crie um modelo causal. Determine quais são os sintomas possíveis, as doenças, os tratamentos e os resultados. Em seguida, desenhe arcos entre eles, indicando que doenças causam cada um dos sintomas e que tratamentos aliviam os sintomas de cada doença. Alguns desses itens serão bem

conhecidos para o especialista do domínio e outros virão da literatura. Com frequência, o modelo combinará bem com as descrições gráficas informais apresentadas em textos da literatura médica.

Simplifique até chegar a um modelo de decisão qualitativa. Tendo em vista que estamos usando o modelo para tomar decisões de tratamento e não para outras finalidades (como determinar a probabilidade conjunta de certas combinações de sintoma/doença), muitas vezes podemos simplificar, removendo variáveis que não estão envolvidas em decisões de tratamento. Às vezes, as variáveis terão de ser divididas ou reunidas para corresponder às intuições do especialista. Por exemplo, o modelo original de estreitamento da aorta tinha uma variável *Tratamento* com valores *cirurgia*, *angioplastia* e *medicação*, e uma variável separada para representar o *Timing* (o melhor momento) do tratamento. Porém, o especialista teve dificuldades para considerar essas variáveis separadamente e, assim, elas foram combinadas, com *Tratamento* assumindo valores como *cirurgia em um mês*. Isso nos dá o modelo da Figura 16.10.

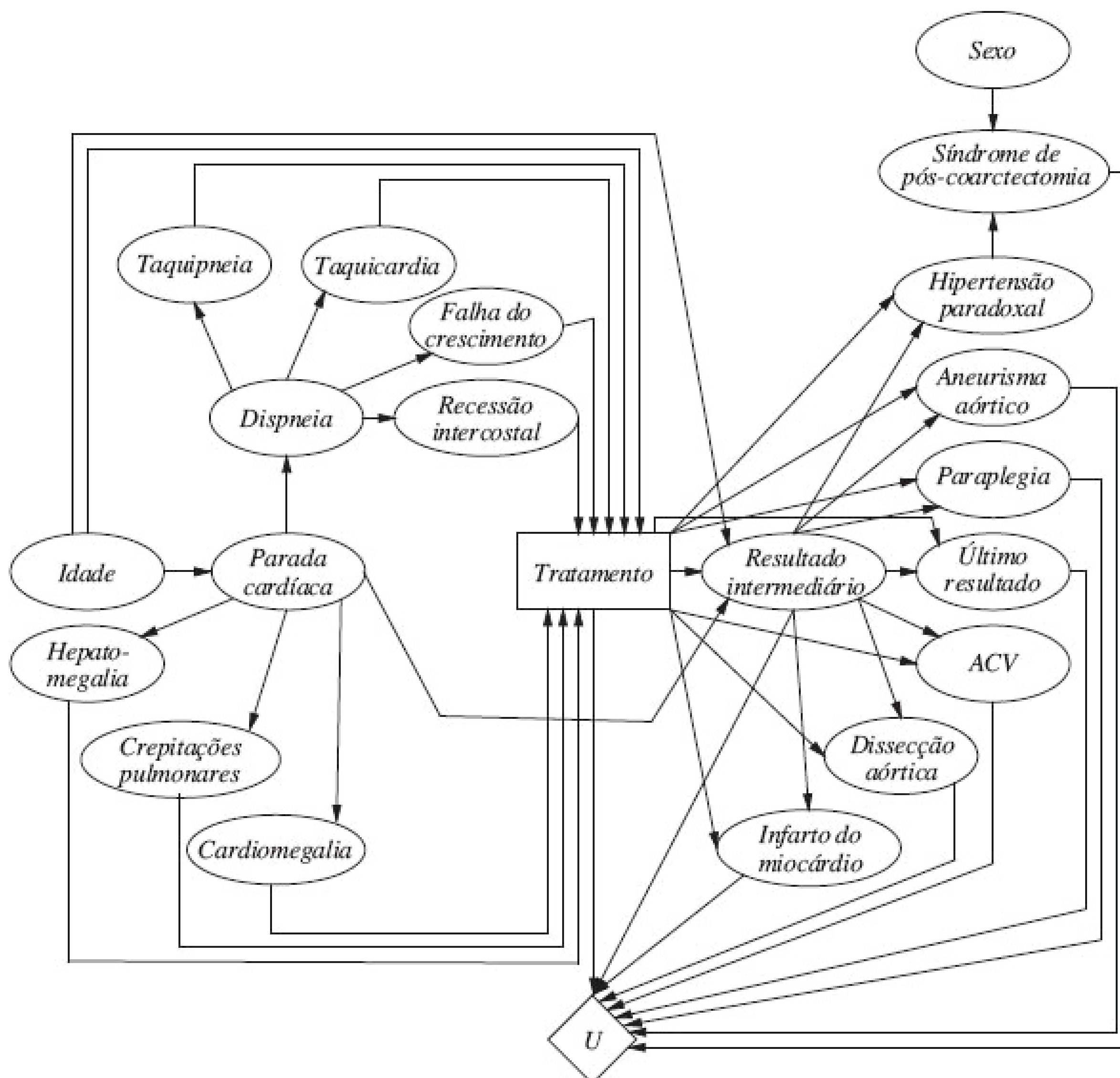


Figura 16.10 Diagrama de influência para estreitamento da aorta (cortesia de Peter Lucas).

Atribua probabilidades. As probabilidades podem vir de bancos de dados de pacientes, de estudos de literatura ou de avaliações subjetivas do especialista. Observe que um sistema de diagnóstico vai concluir, a partir de sintomas e outras observações, a doença ou outra causa dos problemas. Assim, no início dos anos de construção desses sistemas, dado um efeito, era questionado

aos especialistas sobre a probabilidade de determinada causa. Em geral, eles achavam isso difícil de fazer e preferiam avaliar a probabilidade de dado efeito de uma causa. Assim, os sistemas modernos de modo geral avaliam o conhecimento causal e o codificam diretamente na estrutura da rede bayesiana do modelo, deixando o raciocínio do diagnóstico para os algoritmos de inferência da rede bayesiana (Shachter e Heckerman, 1987).

Atribua utilidades. Quando existe um número pequeno de resultados possíveis, eles podem ser enumerados e avaliados individualmente utilizando os métodos da Seção 16.3.1. Criaríamos uma escala desde o melhor até o pior resultado e daríamos um valor numérico a cada um, por exemplo, 0 para morte e 1 para recuperação completa. Em seguida, colocaríamos os outros resultados nessa escala. Isso pode ser feito pelo especialista,mas será melhor se o paciente (ou, no caso de crianças, os pais do paciente) estiver envolvido porque pessoas diferentes têm preferências diferentes. Se houver um número exponencialmente grande de resultados, precisaremos de algum modo de combiná-los usando funções utilidade multiatributo. Por exemplo, podemos dizer que a utilidade negativa de diversas complicações é aditiva.

Verifique e refine o modelo. Para avaliar o sistema, precisaremos de um conjunto de pares (entrada, saída) corretos; um **padrão-ouro**, como ele é chamado, a fim de servir de base de comparação. Para sistemas especialistas médicos, em geral isso significa reunir os melhores médicos disponíveis, apresentar-lhes alguns casos. E pedir-lhes um diagnóstico e um plano de tratamento recomendado para cada caso. Em seguida, observamos o quanto o sistema corresponde a suas recomendações. Se o sistema se sair mal, tentaremos isolar e corrigir as partes que geram resultados errados. Pode ser útil executar o sistema "de trás para a frente". Em vez de apresentar sintomas ao sistema e solicitar um diagnóstico, podemos apresentar um diagnóstico como "falha do coração", examinar a probabilidade prevista de sintomas como taquicardia e compará-la com a literatura médica.

Execute a análise de sensibilidade. Essa importante etapa verifica se a melhor decisão é sensível a pequenas mudanças nas probabilidades e utilidades atribuídas pela variação sistemática desses parâmetros e pela execução repetida da avaliação. Se pequenas mudanças levam a decisões significativamente diferentes, talvez compense gastar mais recursos para reunir dados melhores. Se todas as variações levarem à mesma decisão, o usuário terá mais confiança em que essa é a decisão correta. A análise de sensibilidade é particularmente importante porque uma das principais críticas às abordagens probabilísticas para sistemas especialistas se refere à grande dificuldade para avaliar as probabilidades numéricas exigidas. Com frequência, a análise de sensibilidade revela que muitos números precisam ser especificados apenas de forma muito aproximada. Por exemplo, poderíamos estar inseguros sobre a probabilidade condicional $P(taquicardia \mid dispneia)$, mas, se a decisão ótima for razoavelmente robusta a variações pequenas na probabilidade, a nossa ignorância é uma preocupação menor.

16.8 RESUMO

Este capítulo mostrou como combinar a teoria da utilidade com a probabilidade para permitir a um agente selecionar ações que vão maximizar seu desempenho esperado.

- A **teoria da probabilidade** descreve aquilo em que um agente deve acreditar com base na evidência, a **teoria da utilidade descreve** o que um agente quer e a **teoria da decisão** reúne as outras duas para descrever o que um agente deve fazer.
- Podemos usar a teoria da decisão para construir um sistema que toma decisões considerando todas as ações possíveis e escolhendo aquela que leva ao melhor resultado esperado. Tal sistema é conhecido como **agente racional**.
- A teoria da utilidade mostra que um agente cujas preferências entre loterias são consistentes com um conjunto de axiomas simples pode ser descrito como um agente que possui uma função utilidade.
- A **teoria da utilidade multiatributo** lida com utilidades que dependem de vários atributos distintos de estados. A **dominância estocástica** é uma técnica particularmente útil para a tomada de decisões não ambíguas, mesmo sem valores de utilidade precisos para atributos.
- As **redes de decisão** fornecem um formalismo simples para expressar e resolver problemas de decisão. Elas constituem uma extensão natural de redes bayesianas, contendo nós de decisão e de utilidade, além de nós de acaso.
- Às vezes, a resolução de um problema envolve a descoberta de mais informações antes de tomar uma decisão. O **valor da informação** é definido como a melhora esperada na utilidade, em comparação com a tomada de uma decisão sem a informação.
- Os **sistemas especialistas** que incorporam informações de utilidade têm capacidades adicionais em comparação com sistemas de inferência puros. Além de poder tomar decisões, eles podem utilizar o valor da informação para decidir as perguntas a serem feitas, se houver; podem recomendar planos de contingência e podem calcular a sensibilidade das suas decisões a pequenas mudanças nas avaliações de probabilidade e utilidade.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O livro *L'art de penser*, também conhecido como *Port-Royal Logic* (Arnauld, 1662) afirma:

Para julgar o que é preciso fazer para obter o bem ou evitar o mal, é necessário considerar não só o bem e o mal em si, mas também a probabilidade de que aconteça ou não e visualizar geometricamente a proporção que todas têm em conjunto.

Textos modernos falam de *utilidade* em vez do bem e do mal, mas essa afirmação observa corretamente que se deve multiplicar a utilidade pela probabilidade ("visão geométrica") para dar a utilidade esperada e maximizá-la mais que todos os resultados ("todas essas coisas") para "julgar o que deve ser feito". É notável o quanto isso está certo, há 350 anos, e apenas oito anos depois de Pascal e Fermat mostrarem como usar corretamente a probabilidade. O *Port-Royal Logic* também marcou a primeira publicação da aposta de Pascal.

Daniel Bernoulli (1738), investigando o paradoxo de São Petersburgo (veja o Exercício 16.3), foi o primeiro a perceber a importância da medição de preferências para loterias, escrevendo que "o *valor* de um item não deve se basear em seu *preço*, mas na *utilidade* que ele gera" (os grifos são do autor). O filósofo utilitarista Jeremy Bentham (1823) propôs o **cálculo hedônico** para ponderar

"prazeres" e "dores", argumentando que todas as decisões (não apenas as decisões monetárias) poderiam ser reduzidas a comparações entre utilidades.

A derivação de utilidades numéricas a partir de preferências foi realizada primeiro por Ramsey (1931); os axiomas para preferência neste texto estão mais próximos em forma aos que foram redescobertos na *Theory of Games and Economic Behavior* (von Neumann e Morgenstern, 1944). Uma boa apresentação desses axiomas, no curso de uma discussão sobre preferência de risco, é feita por Howard (1977). Ramsey derivou probabilidades subjetivas (não apenas utilidades) a partir das preferências de um agente; Savage (1954) e Jeffrey (1983) elaboram construções mais recentes desse tipo. Von Winterfeldt e Edwards (1986) fornecem uma perspectiva moderna sobre a análise da decisão e seu relacionamento com as estruturas de preferências humanas. A medida de utilidade micromorte é discutida por Howard (1989). Um levantamento feito em 1994 pelo *Economist* definiu o valor de uma vida entre US$750.000 e US$2,6 milhões. Porém, Richard Thaler (1992) descobriu efeitos irracionais de enquadramento sobre os preços que alguém está disposto a pagar para evitar o risco de morrer, em comparação com o preço que alguém está disposto a receber para aceitar um risco. Para uma chance de 1/1.000, uma pessoa consultada não pagaria mais de $200 para remover o risco, mas não aceitaria $50.000 para assumir o risco. O quanto as pessoas estão dispostas a pagar por um QALY? Quando se trata de um caso específico de salvar a si mesmo ou a um membro da família, o número é aproximadamente "o que eu tiver". Mas podemos perguntar em nível social: suponha que exista uma vacina que produziria X QALYs mas custa Y dólares; vale a pena? Nesse caso, as pessoas relatam ampla gama de valores de cerca de $10.000-150.000 por QALY (Prades *et al*., 2008). Os QALYs são muito mais extensamente usados na tomada de decisões médicas e de política social que as micromortes; veja em Russell (1990) um exemplo típico de argumento para uma mudança importante na política de saúde pública, com base no aumento da utilidade esperada medida em QALYs.

Smith e Winkler (2006) trouxeram de maneira firme a **maldição do otimizador** à vista dos analistas de decisão, que apontaram que os benefícios financeiros projetados para o cliente pelos analistas para o curso da ação proposta quase nunca se materializavam. Eles traçaram isso diretamente a partir da tendência introduzida de selecionar uma ação ideal e mostraram que uma análise bayesiana mais completa elimina o problema. O mesmo conceito subjacente foi chamado de **decepção pós-decisão** por Harrison e March (1984) e foi observado no contexto da análise de projetos de investimento de capital por Brown (1974). A maldição do otimizador está também intimamente relacionada com a maldição do vencedor (Cape *et al*., 1971; Thaler, 1992), que se aplica à licitação em leilões: quem ganha o leilão é muito provável que tenha superestimado o valor do objeto em questão. Cape *et al*. citaram um engenheiro de petróleo no tema de licitação para exploração de direitos de petróleo: "Se alguém ganha um trato contra dois ou três outros pode sentir-se bem sobre a sua boa sorte. Mas como se sentiria se ganhasse contra 50 outras pessoas?" Finalmente, por trás das duas maldições existe o fenômeno geral da **regressão à média**, segundo a qual os indivíduos selecionam com base em características excepcionais do desejo previamente exposto, com alta probabilidade de tornarem-se menos excepcionais no futuro.

O paradoxo de Allais, devido ao ganhador do Prêmio Nobel, o economista Maurice Allais (1953), foi testado experimentalmente (Tversky e Kahneman, 1982; Conlisk, 1989) para mostrar que as pessoas são consistentemente inconsistentes em seus julgamentos. O paradoxo de Ellsberg sobre a

aversão à ambiguidade foi introduzido na tese de Ph.D. de Daniel Ellsberg (Ellsberg, 1962), que passou a ser analista militar da Corporação RAND e vazou documentos conhecidos como The Pentagon Papers, o que contribuiu para o fim da guerra do Vietnã e a renúncia do presidente Nixon. Fox e Tversky (1995) descrevem um estudo mais aprofundado de aversão à ambiguidade. Mark Machina (2005) dá uma visão geral de escolha sob incerteza e como ela pode variar da teoria da utilidade esperada.

Houve uma manifestação recente de livros populares sobre a irracionalidade humana. O mais conhecido é *Predictably Irrational* (Ariely, 2009); outros incluem *Sway* (Brafman e Brafman, 2009), *Nudge* (Thaler e Sunstein, 2009), *Kluge* (Marcus, 2009), *How We Decide* (Lehrer, 2009) e *On Being Certain* (Burton, 2009). Eles complementam o clássico (Kahneman *et al*., 1982) e o artigo que começou isso tudo (Kahneman e Tversky, 1979). O campo da psicologia evolutiva (Buss, 2005), por outro lado, é contrário a essa literatura, argumentando que os seres humanos são muito racionais em contextos evolutivamente adequados. Seus adeptos apontam que a irracionalidade é penalizada, por definição, em um contexto evolutivo e mostra que, em alguns casos, é um artefato de configuração experimental (Cummins e Allen, 1998). Houve ressurgimento recente do interesse em modelos bayesianos de cognição, derrubando décadas de pessimismo (Oaksford e Chater, 1998; Elio, 2002; Chater e Oaksford, 2008).

Keeney e Raiffa (1976) fornecem uma introdução completa à teoria da utilidade multiatributo. Eles descrevem as primeiras implementações de computador de métodos para extrair os parâmetros necessários a uma função utilidade multiatributo e incluem uma extensiva relação de aplicações reais da teoria. Em IA, a principal referência para MAUT é o trabalho de Wellman (1985), que inclui um sistema chamado URP (Utility Reasoning Package), que pode usar uma coleção de declarações sobre a independência de preferências e a independência condicional para analisar a estrutura de problemas de decisão. O uso da dominância estocástica, juntamente com modelos de probabilidade qualitativos, foi extensamente investigado por Wellman (1988, 1990a). Wellman e Doyle (1992) fornecem um esboço preliminar de como um conjunto complexo de relacionamentos de independência de utilidade poderia ser utilizado para fornecer um modelo estruturado de uma função utilidade, de modo muito semelhante à forma como as redes bayesianas fornecem um modelo estruturado de distribuições de probabilidade conjunta. Bacchus e Grove (1995, 1996) e também La Mura e Shoham (1999) apresentam resultados adicionais que seguem essas linhas.

A teoria da decisão tem sido uma ferramenta padrão em economia, finanças e gestão desde a década de 1950. Até os anos 1980, as árvores de decisão eram a principal ferramenta utilizada para representar problemas de decisão simples. Smith (1988) fornece uma visão geral da *metodologia* da análise da decisão. As redes de decisão ou diagramas de influência foram introduzidas por Howard e Matheson (1984), com base em trabalho anterior no SRI (Miller *et al*., 1976). O método de Howard e Matheson envolvia a derivação de uma árvore de decisão a partir de uma rede de decisão, mas, em geral, a árvore tem tamanho exponencial. Shachter (1986) desenvolveu um método para tomada de decisões baseado diretamente em uma rede de decisão, sem a criação de uma árvore de decisão intermediária. Esse algoritmo também foi um dos primeiros a fornecer inferência completa para redes bayesianas com várias conexões. Zhang *et al.* (1994) mostraram como tirar vantagem da independência condicional de informação para reduzir o tamanho das árvores na prática; eles utilizaram a expressão *rede de decisão* para redes que usam essa abordagem (embora outros a

utilizem como sinônimo de diagrama de influência). O trabalho recente de Nilsson e Lauritzen (2000) vincula algoritmos para redes de decisão a desenvolvimentos contínuos em algoritmos de formação de agrupamentos para redes bayesianas. Koller e Milch (2003) mostram como os diagramas de influência podem ser usados para resolver jogos que envolvem a coleta de informações por jogadores adversários, e Detwarasiti e Shachter (2005) mostram como diagramas de influência podem ser usados como auxílio para a tomada de decisão por uma equipe que compartilha objetivo mas é incapaz de compartilhar todas as informações perfeitamente. A coleção de Oliver e Smith (1990) tem vários artigos úteis sobre redes de decisão, como também o número especial de 1990 do periódico *Networks*. Os artigos sobre redes de decisão e modelagem de utilidade também aparecem regularmente nos periódicos *Management Science* e *Decision Analysis*.

A teoria do valor da informação foi explorada em primeiro lugar no contexto de experimentos estatísticos, onde foi utilizada uma quase utilidade (redução da entropia) (Lindley, 1956). O teórico de controle russo Ruslan Stratonovich (1965) desenvolveu a teoria mais geral aqui apresentada, em que a informação tem valor em virtude de sua capacidade de afetar decisões. O trabalho de Stratonovich não era conhecido no Ocidente, onde Ron Howard (1966) foi pioneiro com a mesma ideia. Seu papel termina com a observação: "Se a teoria da informação de valor e as estruturas teóricas associadas à decisão no futuro não ocuparem grande parte da educação de engenheiros, a profissão de engenharia vai achar que seu papel tradicional de gestão dos recursos científicos e econômicos para o benefício do homem foi perdida para outra profissão." Até o momento, a revolução que implica métodos de gestão não ocorreu.

O trabalho recente de Krause e Guestrin (2009) mostra que o cálculo do valor não míope exato da informação é intratável, mesmo em redes de múltiplas árvores. Há outros casos — mais restritos do que o valor geral de informações — em que o algoritmo míope prevê uma aproximação comprovadamente boa para a sequência ótima de observações (Krause *et al*., 2008). Em alguns casos — por exemplo, à procura de um tesouro enterrado em um dos *n* lugares —, experimentos em ordem de probabilidade de sucesso dividido pelo custo fornecem uma solução ótima (Kadane e Simon, 1977).

Surpreendentemente, poucos pesquisadores de IA adotaram ferramentas de teoria da decisão depois das primeiras aplicações em tomada de decisões médicas descritas no Capítulo 13. Uma das poucas exceções foi Jerry Feldman, que aplicou a teoria da decisão a problemas da visão (Feldman e Yakimovsky, 1974) e de planejamento (Feldman e Sproull, 1977). Depois do ressurgimento do interesse em métodos probabilísticos em IA, na década de 1980, os sistemas especialistas de teoria da decisão ganharam ampla aceitação (Horvitz *et al*.; Cowell *et al*., 2002). De fato, de 1991 em diante, o projeto de capa do periódico *Artificial Intelligence* passou a representar uma rede de decisão, embora pareça ter sido adotada alguma licença artística na orientação das setas.

EXERCÍCIOS

16.1 (Adaptado de David Heckerman.) Este exercício se refere ao **Almanac Game**, utilizado por analistas de decisões para calibrar estimativas numéricas. Para cada uma das perguntas que seguem, forneça seu melhor palpite sobre a resposta, isto é, um número que você imagina ter a mesma

probabilidade de ser muito alto quanto de ser muito baixo. Também dê seu palpite em uma estimativa de 25º percentil, ou seja, um número que você imagina ter uma chance de 25% de ser muito alto e uma chance de 75% de ser muito baixo. Faça o mesmo para o 75º percentil. (Desse modo, você deve fornecer três estimativas ao todo — baixa, mediana e alta — para cada pergunta.)

a. Número de passageiros que voaram entre Nova York e Los Angeles em 1989.

b. População de Varsóvia em 1992.

c. Ano em que Coronado descobriu o rio Mississippi.

d. Número de votos recebidos por Jimmy Carter na eleição presidencial de 1976.

e. Idade da árvore viva mais antiga em 2002.

f. A altura da represa Hoover em pés.

g. Número de ovos produzidos no Oregon em 1985.

h. Número de budistas no mundo em 1992.

i. Número de mortes por aids nos Estados Unidos em 1981.

j. Número de patentes concedidas nos Estados Unidos em 1901.

As respostas corretas aparecem depois do último exercício deste capítulo. A partir do ponto de vista da análise da decisão, o detalhe interessante não é o quanto seus palpites medianos se aproximaram das respostas reais, mas a frequência com que a resposta real ficou dentro de seus limites de 25% e 75%. Se essa frequência foi de aproximadamente metade do tempo, seus limites são precisos. Porém, se for como a maioria das pessoas, você estará mais seguro de si do que deveria estar porque, nesse caso, menos da metade das respostas ficará dentro dos limites. Com a prática, você poderá ajustar suas respostas a limites realistas e, desse modo, será mais útil para fornecer informações que serão usadas na tomada de decisões. Tente este segundo conjunto de perguntas e veja se há alguma melhora:

a. Ano de nascimento de Zsa Zsa Gabor.

b. Distância máxima de Marte ao Sol em milhas.

c. Valor em dólares das exportações de trigo dos Estados Unidos em 1992.

d. Toneladas de mercadorias manipuladas no Porto de Honolulu em 1991.

e. Salário anual em dólares do governador da Califórnia em 1993.

f. População de San Diego em 1990.

g. Ano em que Roger Williams fundou Providence, em Rhode Island.

h. Altura do monte Kilimanjaro em pés.

i. Extensão da Ponte de Brooklyn em pés.

j. Número de mortes em acidentes automobilísticos nos Estados Unidos em 1992.

16.2 Chris considera cinco carros usados antes de comprar um com utilidade máxima esperada. Pat considera 11 carros e faz o mesmo. Considerando o restante equivalente, quem provavelmente terá o melhor carro? Quem tem mais probabilidade de ficar decepcionado com a qualidade do carro? Por quanto (em termos do desvio-padrão da qualidade esperada)?

16.3 Em1713, Nicolas Bernoulli expôs um quebra-cabeças, hoje chamado de paradoxo de São Petersburgo, que funciona assim: você tem a oportunidade de participar de um jogo em que uma moeda imparcial é lançada repetidamente até dar o resultado cara. Se o primeiro resultado cara aparecer no n-ésimo lançamento, você ganha 2^n dólares.

a. Mostre que o valor monetário esperado desse jogo é infinito.

b. Quanto você pessoalmente pagaria para participar do jogo?

c. O primo de Nicolas, Daniel Bernoulli, resolveu o aparente paradoxo em 1738 sugerindo que a utilidade do dinheiro é medida em uma escala logarítmica (isto é, $U(S_n) = a \log_2 n + b$, onde S_n é o estado de ter \$$n$). Qual é a utilidade esperada do jogo sob essa hipótese?

d. Qual quantia máxima seria racional pagar para participar do jogo, supondo-se que a riqueza inicial de alguém seja \$$k$?

16.4 Escreva um programa de computador para automatizar o processo no Exercício 16.9. Experimente seu programa com diversas pessoas de diferentes perspectivas e visões políticas. Comente a consistência de seus resultados, tanto para um indivíduo quanto para diversos indivíduos.

16.5 A Companhia de Doces Surpresa fabrica doces de dois sabores: 70% são de sabor morango e 30% são de sabor anchova. Cada novo pedaço de doce começa com uma forma redonda; à medida que se move ao longo da linha de produção, uma máquina seleciona aleatoriamente determinada porcentagem a ser aparada em um quadrado; então, cada peça é acondicionada em uma embalagem cuja cor é escolhida aleatoriamente, podendo ser vermelho ou marrom. Oitenta por cento dos doces de morango são redondos e 80% têm embalagem vermelha, enquanto 90% dos doces de anchova são quadrados e 90% têm embalagem marrom. Todos os doces são vendidos individualmente em caixas pretas, lacradas e idênticas.

Agora você, o cliente, acabou de comprar um doce Surpresa na loja, mas ainda não abriu a caixa. Considere as três redes de Bayes na Figura 16.11.

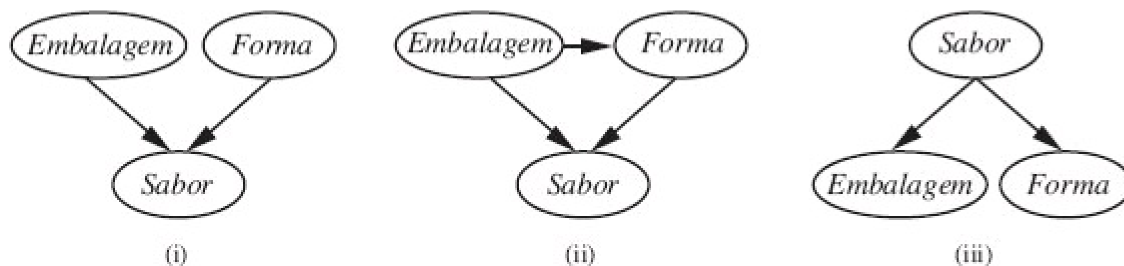


Figura 16.11 Três redes de Bayes propostas para o problema Doce Surpresa, Exercício 16.5.

a. Que rede pode representar corretamente **P**(*Sabor*, *Embalagem*, *Forma*)?

b. Qual rede é a melhor representação para esse problema?

c. A rede (i) afirma que **P**(*Embalagem* | *Forma*) = **P**(*Embalagem*)?

d. Qual é a probabilidade de que seu doce tenha embalagem vermelha?

e. Na caixa há um doce redondo com embalagem vermelha. Qual a probabilidade de que seu sabor seja morango?

f. Um doce de morango desembrulhado vale s no mercado aberto e um doce de anchova

desembrulhado vale a. Escreva uma expressão para o valor de uma caixa de doces fechada.

g. Uma nova lei proíbe o comércio de doces desembrulhados, mas ainda é legal vender doces na embalagem (fora da caixa). Agora uma caixa de doces fechado vale mais ou menos, ou o mesmo do que antes?

16.6 Demonstre que os julgamentos $B \succ A$ e $C \succ D$ no paradoxo de Allais violam o axioma da substitutibilidade.

16.7 Considere o paradoxo de Allais: um agente que prefere B sobre A (fazendo tudo certo) e C sobre D (escolhendo o maior VME) não está agindo racionalmente, de acordo com a teoria da utilidade. Você acha que isso indica um problema para o agente, um problema para a teoria ou nenhum problema? Explique.

16.8 Os bilhetes de uma loteria custam \$1. Há dois prêmios possíveis: pagamento de \$10 com probabilidade de 1/50 e o pagamento de \$1.000.000 com probabilidade de 1/2.000.000. Qual seria o valor monetário esperado de um bilhete de loteria? Quando (se em algum caso) é racional comprar um bilhete de loteria? Seja preciso – mostre uma equação que envolva utilidades. Você pode assumir a riqueza atual de $\$k$ e que $U(S_k) = 0$. Assuma também que $U(S_{k+10}) = 10 \times U(S_{k+1})$, mas não faça nenhuma suposição em relação a $U(S_{k+1.000.000})$. Estudos sociológicos mostram que as pessoas com menos renda compram uma quantidade desproporcional de bilhetes de loteria. Você considera que isso se deve a eles não serem bons tomadores de decisão ou porque possuem uma função utilidade diferente? Considere o valor de contemplar a possibilidade de ganhar na loteria versus o valor de ser contemplado em se tornar um herói de ação ao assistir a um filme de aventura.

16.9 Avalie sua própria utilidade para quantidades incrementais diferentes de dinheiro, executando uma série de testes de preferência entre alguma quantia definida M_1 e uma loteria $[p, M_2; (1-p), 0]$. Escolha valores diferentes de M_1 e M_2, e faça p variar até ficar indiferente entre as duas escolhas. Represente a função utilidade resultante.

16.10 Quanto vale para você uma micromorte? Crie um protocolo para definir isso. Faça perguntas baseadas no pagamento para evitar o risco e a possibilidade de receber para aceitar o risco.

16.11 Considere as variáveis contínuas $X_1, \ldots, X_k$ distribuídas independentemente de acordo com a mesma função de densidade de probabilidade $f(x)$. Demonstre que a função de densidade para $\max\{X_1, \ldots, X_k\}$ é dada por $kf(x)\,(F(x))^{k-1}$, onde F é a distribuição cumulativa de f.

16.12 Os economistas muitas vezes fazem uso de uma função utilidade exponencial para o dinheiro: $U(x) = -e^{x/R}$, onde R é uma constante positiva que representa a tolerância ao risco do indivíduo. A tolerância ao risco reflete a probabilidade de um indivíduo aceitar uma loteria com valor monetário esperado em particular (VME) *versus* algum retorno certo. Como R (que é medido na mesma unidade que x) torna-se maior, o indivíduo torna-se menos avesso ao risco.

a. Assuma que Maria tem uma função utilidade exponencial com $R = \$500$. Maria pode escolher entre receber \$500 com certeza (probabilidade 1) ou participar de uma loteria que tenha probabilidade de 60% de ganhar \$5.000 e probabilidade de 40% de não ganhar nada. Assumindo que Maria age racionalmente, qual opção ela vai escolher? Mostre como você derivou sua resposta.

b. Considere a escolha entre ter certeza de receber $100 (probabilidade 1) ou participar de uma loteria que tenha probabilidade de 50% de ganhar e 50% de não ganhar nada. O valor aproximado de *R* (com três dígitos significativos) em uma função utilidade exponencial faria com que um indivíduo fosse indiferente a essas duas alternativas (pode ser útil codificar um programa pequeno para ajudar a resolver esse problema).

16.13 Repita o Exercício 16.16 usando a representação de ação-utilidade mostrada na Figura 16.7.

16.14 Para qualquer dos diagramas de localização do aeroporto dos Exercícios 16.16 e 16.13, a qual item da tabela de probabilidade condicional a utilidade é mais sensível, dada a evidência disponível?

16.15 Considere um aluno que tenha a opção de comprar ou não um livro para um curso. Modelaremos isso como um problema de decisão com um nó de decisão booleano, *B*, indicando se o agente optou por comprar o livro, e dois nós de acaso booleanos, *M*, indicando se o aluno dominou o material no livro, e *P*, indicando se o aluno passou no curso. Claro, há também um nó de utilidade, *U*. Determinado aluno, Sam, tem uma função utilidade aditiva: 0 para não comprar o livro e –$100 para comprá-lo; $2.000 para passar no curso e 0 para não passar. Segue as estimativas de probabilidade condicionais de Sam:

$P(p \mid b, m) = 0{,}9$

$P(p \mid b, \neg m) = 0{,}5$

$P(p \mid \neg b, m) = 0{,}8$

$P(p \mid \neg b, \neg m) = 0{,}3$

$P(m \mid b) = 0{,}9$

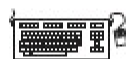
$P(m \mid \neg b) = 0{,}7$

Você pode pensar que *P* seria independente de *B* dado *M*, mas nese curso, ao final, pode-se utilizar o livro; assim, ter o livro ajuda.

a. Desenhe a rede de decisão para esse problema.

b. Calcule a utilidade esperada de comprar o livro e de não comprá-lo.

c. O que Sam deve fazer?

**16.16** Este exercício completa a análise do problema de localização do aeroporto da Figura 16.6.

a. Forneça domínios de variáveis, probabilidades e utilidades razoáveis para a rede supondo que existam três locais possíveis.

b. Resolva o problema de decisão.

c. O que acontecerá se as mudanças na tecnologia indicarem que cada aeronave gera metade do ruído?

d. E se evitar o ruído se tornar três vezes mais importante?

e. Calcule o VIP para *TráfegoAéreo*, *Litígio* e *Construção* em seu modelo.

16.17 (Adaptado de Pearl (1988).) Um comprador de carros usados pode decidir realizar vários

testes com diversos custos (por exemplo, chutar os pneus, levar o carro a um mecânico qualificado) e depois, dependendo do resultado dos testes, decidir que carro comprar. Vamos supor que o comprador esteja decidindo comprar o carro c_1, que exista tempo para executar no máximo um teste e que t_1 é o teste de c_1 e custa \$50.

Um carro pode estar em bom estado (qualidade θ^+) ou em mau estado (qualidade q–) e os testes podem ajudar a indicar em que estado o carro se encontra. O carro c_1 custa \$1.500 e seu valor de mercado é \$2.000, se estiver em bom estado; caso contrário, serão necessários \$700 em reparos para colocá-lo em boas condições. A estimativa do comprador é que c_1 tem uma chance de 70% de se encontrar em bom estado.

a. Desenhe a rede de decisão que representa esse problema.

b. Calcule o ganho líquido esperado da compra de c_1 sem a realização de qualquer teste.

c. Os testes podem ser descritos pela probabilidade de o carro ser aprovado (passar) ou ser reprovado (não passar) no teste, dado que o carro está em bom estado ou mau estado. Temos as seguintes informações:

$P(passar(c_1, t_1)|\theta+ (c_1)) = 0{,}8$

$P(passar(c_1,t_1)|q- (c_1)) = 0{,}35$

Use o teorema de Bayes para calcular a probabilidade de o carro passar (ou ser reprovado) em seu teste e, consequentemente, a probabilidade de que ele esteja em bom (ou mau) estado, dado cada resultado de teste possível.

d. Calcule as decisões ótimas dada uma aprovação ou reprovação no teste e suas utilidades esperadas.

e. Calcule o valor da informação do teste e derive um plano condicional ótimo para o comprador.

16.18 Lembre-se da definição do *valor da informação* na Seção 16.6.

a. Demonstre que o valor da informação é não negativo e independente de ordem.

b. Explique por que algumas pessoas preferem não obter informação — por exemplo, não querer saber o sexo do bebê quando é feito um ultrassom.

c. Uma função f sobre conjuntos é **submodular** se, para qualquer elemento de x e quaisquer conjuntos A e B tal que $A \subseteq B$ acrescentar x a A gera maior aumento em f que adicionar x a B:

$$A \subseteq B \Rightarrow (f(A \cup \{x\}) - f(A)) \geq (f(B \cup \{x\}) - f(B)) .$$

A submodularidade capta a noção intuitiva de *retornos decrescentes*. O valor da informação é visto como uma função f sobre conjuntos de observações possíveis, submodulares? Demonstre ou encontre um contraexemplo.

As respostas do Exercício 16.1 (onde M representa um milhão) são: primeiro conjunto: 3M, 1,6M, 1.541, 41M, 4.768, 221, 649M, 295M, 132, 25.546; segundo conjunto: 1.917, 155M, 4.500M, 11M, 120.000, 1,1M, 1.636, 19.340, 1.595, 41.710.

[1] A teoria da decisão clássica deixa o estado atual S_0 implícito, mas podemos torná-lo explícito, escrevendo $P(\text{RESULTADO }(a) = s' \mid a, \mathbf{e})\ \Sigma_s\ P(\text{RESULTADO }(s, a) = s' \mid a)P(S_0 = s \mid \mathbf{e})$.

[2] Pedimos desculpas aos leitores cujas companhias aéreas locais não mais oferecem refeições em voos longos.

[3] Podemos levar em conta o prazer de jogar codificando eventos de jogos na descrição do estado; por exemplo, "Tem 10 reais e jogou" talvez fosse preferível a "Tem 10 reais e não jogou".

[4] Nesse sentido, utilidades assemelham-se às temperaturas: a temperatura em Fahrenheit é 1,8 vez a temperatura em graus Celsius mais 32. Você obtém os mesmos resultados em qualquer sistema de medição.

[5] Tal comportamento poderia ser chamado de desesperado, mas é racional se alguém já está em situação desesperada.

[6] Por exemplo, o matemático/mágico Persi Diaconis pode fazer o lançamento de uma moeda sair do jeito que ele quer todas as vezes (Landhuis, 2004).

[7] Mesmo o que é certo pode não ser seguro. Apesar de promessas fortes, ainda não recebemos os 27 milhões de dólares da conta bancária nigeriana de um parente falecido até então desconhecido.

[8] Em alguns casos, talvez seja necessário subdividir o intervalo de valores de tal forma que a utilidade varie monotonicamente dentro de cada intervalo. Por exemplo, se o atributo *TemperaturaSala* tiver um pico de utilidade a 21°C, o dividiremos em dois atributos medindo a diferença do ideal, um mais frio e outro mais quente. A utilidade seria, então, monotonicamente crescente em cada atributo.

[9] Esses nós são chamados **nós de valor** na literatura.

[10] Nos Estados Unidos, a única pergunta que sempre é feita de antemão é se o paciente tem plano de saúde.

[11] Não há perda de expressividade em exigir informação perfeita. Suponha que quiséssemos modelar o caso em que nos tornamos um pouco mais certos sobre uma variável. Podemos fazer isso através da introdução de outra variável sobre a qual aprendemos a informação perfeita. Por exemplo, suponha que inicialmente tenhamos ampla incerteza sobre a variável *temperatura*. Então, obtemos o conhecimento perfeito *Termômetro* = 37; isso nos dá uma informação imperfeita sobre a *Temperatura* verdadeira e a incerteza devida ao erro de medição estar codificado no modelo de sensor $\mathbf{P}(Termômetro \mid Temperatura)$. Consulte o Exercício 16.17 para outro exemplo.

CAPÍTULO 17

Tomada de decisões complexas

Em que examinamos métodos para decidir o que fazer hoje, dado que podemos decidir novamente amanhã.

Neste capítulo, abordaremos as questões computacionais envolvidas na tomada de decisões em ambiente estocástico. Enquanto o Capítulo 16 estava preocupado com problemas de decisão instantânea ou episódica, em que a utilidade do resultado de cada ação era bem conhecida, aqui vamos nos preocupar com **problemas de decisão sequencial**, em que a utilidade do agente depende de uma sequência de decisões. Problemas de decisão sequencial incorporam utilidades, incerteza e percepção, e incluem os problemas de busca e planejamento como casos especiais. A Seção 17.1 explica como os problemas de decisão sequencial são definidos, e as Seções 17.2 e 17.3 explicam como eles podem ser resolvidos para gerar um comportamento ótimo que equilibre os riscos e as recompensas de agir em um ambiente incerto. A Seção 17.4 estende essas ideias ao caso de ambientes parcialmente observáveis, e a Seção 17.4.3 desenvolve um projeto completo para agentes de teoria da decisão em ambientes parcialmente observáveis, combinando as redes bayesianas dinâmicas do Capítulo 15 com as redes de decisão do Capítulo 16.

A segunda parte do capítulo cobre ambientes com múltiplos agentes. Em tais ambientes, a noção de comportamento ótimo se torna muito mais complicada pelas interações entre os agentes. A Seção 17.5 introduz as principais ideias da **teoria dos jogos**, inclusive a ideia de que agentes racionais talvez precisem se comportar de modo aleatório. A Seção 17.6 examina como os sistemas multiagente podem ser projetados para que vários agentes possam alcançar um objetivo comum.

17.1 PROBLEMAS DE DECISÃO SEQUENCIAL

Suponha que um agente esteja situado no ambiente 4 × 3 mostrado na Figura 17.1(a). Começando no estado inicial, ele deve escolher uma ação em cada passo de tempo. A interação com o ambiente termina quando o agente alcança um dos estados objetivos, marcados com +1 ou –1. Assim como para problemas de busca, as ações disponíveis para o agente em cada estado são dadas por AÇÕES(s), algumas vezes abreviado como $A(s)$; no ambiente 4 × 3, as ações em todos os estados são *Acima*, *Abaixo*, *Esquerda* e *Direita*.Vamos supor, por enquanto, que o ambiente seja **completamente observável**, de forma que o agente sempre saiba onde está.

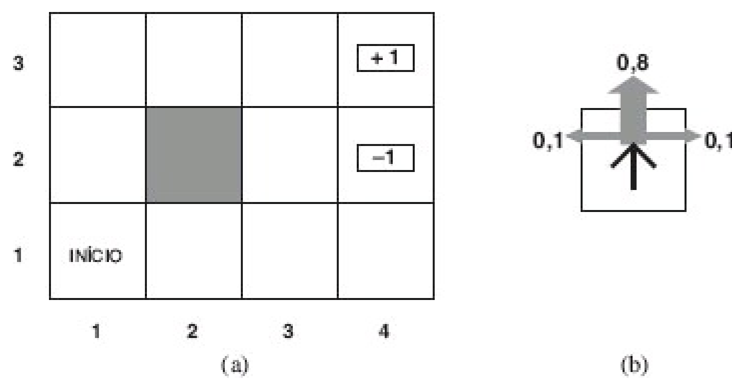


Figura 17.1 (a) Um ambiente simples de 4 × 3 que apresenta ao agente um problema de decisão sequencial. (b) Ilustração do modelo de transição do ambiente: o resultado "pretendido" ocorre com probabilidade 0,8, mas com probabilidade 0,2 o agente se move em um ângulo reto em relação à direção pretendida. Uma colisão com uma parede resulta em nenhum movimento. Os dois estados terminais têm recompensa +1 e –1, respectivamente, e todos os outros estados têm recompensa –0,04.

Se o ambiente fosse determinístico, seria fácil encontrar uma solução: [*Acima*, *Acima*, *Direita*, *Direita*, *Direita*]. Infelizmente, o ambiente nem sempre responderá como esperado com essa solução porque as ações são pouco confiáveis. O modelo específico de movimento estocástico que adotamos está ilustrado na Figura 17.1(b). Cada ação alcança o efeito pretendido com probabilidade 0,8, mas, no restante do tempo, a ação move o agente em ângulos retos até a direção pretendida. Além disso, se o agente bater em uma parede, ele permanecerá no mesmo quadrado. Por exemplo, a partir do quadrado inicial (1,1), a ação *Acima* move o agente para (1,2) com probabilidade 0,8, mas, com probabilidade 0,1, ele se move para a direita até (2,1) e, com probabilidade 0,1, ele se move para a esquerda, choca-se com a parede e fica em (1,1). Em tal ambiente, a sequência [*Acima*, *Acima*, *Direita*, *Direita*, *Direita*] contorna a barreira e alcança o estado de meta em (4,3) com probabilidade $0{,}8^5 = 0{,}32768$. Também existe uma pequena chance de atingir acidentalmente a meta indo por outro caminho, com probabilidade $0{,}1^4 \times 0{,}8$, dando um total geral igual a 0,32776 (veja também o Exercício 17.1).

Como no Capítulo 3, o **modelo de transição** (ou apenas "modelo", quando não gerar confusão) descreve o resultado de cada ação em cada estado. Aqui, o resultado é estocástico, então escrevemos $P(s' \mid s, a)$ para indicar a probabilidade de alcançar o estado s' se a ação a for feita no estado s. Vamos supor que as transições são de **markesianas** no sentido do Capítulo 15, isto é, a probabilidade de alcançar s' a partir de s depende apenas de s, e não do histórico de estados anteriores. No momento, você pode pensar em $P(s' \mid s, a)$ como uma grande tabela tridimensional contendo probabilidades. Mais adiante, na Seção 17.4.3, veremos que o modelo de transição pode ser representado como uma **rede bayesiana dinâmica**, da mesma maneira que no Capítulo15.

Para completar a definição do ambiente de tarefa, devemos especificar a função utilidade para o agente. Como o problema de decisão é sequencial, a função utilidade dependerá de uma sequência de estados — um **histórico do ambiente** —, em vez de depender de um único estado. Mais adiante, nesta seção, investigaremos como tais funções utilidade podem ser especificadas em geral; por enquanto, vamos simplesmente estipular que, em cada estado s, o agente recebe uma **recompensa** $R(s)$, que pode ser positiva ou negativa, mas deve ser limitada. Para nosso exemplo específico, a recompensa é –0,04 em todos os estados, exceto os estados terminais (que têm recompensas +1 e –1). A utilidade de um histórico do ambiente é simplesmente (por enquanto) a *soma* das recompensas

recebidas. Por exemplo, se o agente alcançar o estado +1 depois de 10 passos, sua utilidade total será 0,6. A recompensa negativa igual a –0,04 dá ao agente um incentivo para alcançar (4,3) depressa e, assim, nosso ambiente é uma generalização estocástica dos problemas de busca do Capítulo 3. Outro modo de dizer isso é afirmar que o agente não aprecia viver nesse ambiente e, portanto, quer deixá-lo assim que possível.

Para resumir: um problema de decisão sequencial para um ambiente completamente observável, estocástico, com um modelo de transição de Markov e recompensas aditivas, é chamado de **processo de decisão de Marko**v ou **MDP** (Makov Decison Process), e consiste de um conjunto de estados (com estado inicial s_0); um conjunto de AÇÕES(s) de ações aplicáveis em cada estado; um modelo de transição $P(s' \mid s, a)$ e uma função de recompensa $R(s)$.[1]

A próxima questão é definir qual seria a aparência de uma solução para o problema. Vimos que qualquer sequência fixa de ações não resolverá o problema porque o agente poderia acabar em um estado diferente da meta. Desta forma, uma solução tem de especificar o que o agente deve fazer para *qualquer* estado que o agente possa alcançar. Uma solução desse tipo é chamada de **política**. Normalmente, denotamos uma política por π e π(s) é a ação recomendada pela política π para o estado s. Se o agente tiver uma política completa, não importará o resultado de qualquer ação, o agente sempre saberá o que fazer em seguida.

Toda vez que uma dada política for executada a partir do estado inicial, a natureza estocástica do ambiente poderá levar a um histórico de ambiente diferente. A qualidade de uma política é, portanto, medida pela utilidade *esperada* dos históricos de ambientes possíveis gerados por essa política. Uma **política ótima** é uma política que produz a utilidade esperada mais alta. Usamos π* para denotar uma política ótima. Dado π*, o agente decide o que fazer consultando sua percepção atual, que informa o estado atual s, e depois executando a ação π*(s). Uma política representa explicitamente a função do agente e, portanto, é uma descrição de um agente reflexivo simples, calculada a partir das informações usadas por um agente baseado na utilidade.

Uma política ótima para o mundo da Figura 17.1 é mostrada na Figura 17.2(a). Observe que, como o custo de dar um passo é bastante pequeno em comparação com a penalidade por terminar em (4,2) por acidente, a política ótima para o estado (3,1) é conservadora. A política recomenda seguir o caminho longo, em vez de tomar o atalho e se arriscar a entrar em (4,2).

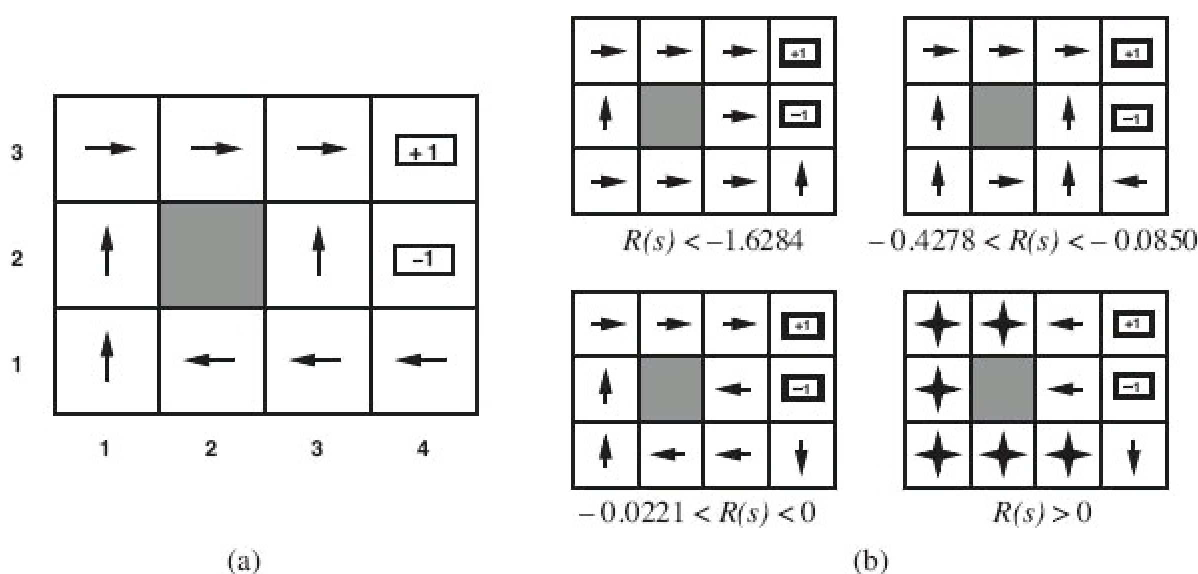


Figura 17.2 (a) Uma política ótima para o ambiente estocástico com $R(s) = –0{,}04$ nos estados não terminais. (b) Políticas ótimas para quatro intervalos diferentes de $R(s)$.

O equilíbrio entre risco e recompensa muda dependendo do valor de $R(s)$ para os estados não terminais. A Figura 17.2(b) mostra políticas ótimas para quatro intervalos diferentes de $R(s)$. Quando $R(s) \leq –1{,}6284$, a vida é tão difícil que o agente vai direto para a saída mais próxima, ainda que a saída tenha o valor –1. Quando $–0{,}4278 \leq R(s) \leq –0{,}0850$, a vida é bastante desagradável; o agente toma a rota mais curta até o estado +1 e está disposto a correr o risco de cair no estado –1 por acidente. Em particular, o agente toma o atalho a partir de (3,1). Quando a vida é apenas ligeiramente ruim ($–0{,}0221 < R(s) < 0$), a política ótima não assume *absolutamente nenhum risco*. Em (4,1) e (3,2) o agente segue diretamente para fora do estado –1, de forma que não possa cair nesse estado por acidente, embora isso signifique bater a cabeça contra a parede várias vezes. Finalmente, se $R(s) > 0$, a vida positivamente é agradável e o agente evita *ambas* as saídas. Desde que as ações em (4,1), (3,2) e (3,3) sejam as que estão representadas, toda política é ótima, e o agente obtém recompensa total infinita porque nunca entra em estado terminal. Surpreendentemente, verificamos que existem seis outras políticas ótimas para vários intervalos de $R(s)$; o Exercício 17.5 pede para encontrá-las.

O equilíbrio cuidadoso entre risco e recompensa é uma característica dos MDPs que não surge em problemas de busca determinística; além disso, é uma característica de muitos problemas de decisão do mundo real. Por essa razão, os MDPs foram estudados em vários campos, inclusive em IA, pesquisa operacional, economia e teoria de controle. Foram propostas dezenas de algoritmos para calcular políticas ótimas. Nas Seções 17.2 e 17.3, descrevemos duas das famílias mais importantes de algoritmos. Porém, primeiro devemos completar nossa investigação das utilidades e políticas para problemas de decisão sequencial.

17.1.1 Utilidades ao longo do tempo

No exemplo de MDP da Figura 17.1, o desempenho do agente foi medido por uma soma de recompensas para os estados visitados. Essa escolha de medida de desempenho não é arbitrária, mas não é a única possibilidade da função utilidade sobre históricos de ambiente, que escrevemos como $U_h([s_0, s_1,\ldots, S_n])$. Nossa análise baseia-se na **teoria da utilidade de multiatributo** (Seção 16.4) e é um pouco técnica; o leitor impaciente pode pular para a próxima seção.

A primeira pergunta a responder é se existe um **horizonte finito** ou um **horizonte infinito** para tomada de decisão. Um horizonte finito significa que existe um tempo *fixo N* depois do qual nada importa — o jogo acabou, por assim dizer. Desse modo, $U_h([s_0, s_1, \ldots, s_{N+k}]) = U_h([s_0, s_1, \ldots, s_n])$ para todo $k > 0$. Por exemplo, suponha que um agente comece em (3,1) no mundo 4 × 3 da Figura 17, e suponha que $N = 3$. Então, para ter qualquer chance de alcançar o estado +1, o agente deve ir diretamente para ele, e a ação ótima é ir *Acima*. Por outro lado, se $N = 100$, existe bastante tempo para seguir a rota segura dirigindo-se para a *Esquerda*. *Assim, com um horizonte finito, a ação ótima em um dado estado poderia mudar com o passar do tempo*. Dizemos que a política ótima para um horizonte finito é **não estacionária**. Por outro lado, sem um limite de tempo fixo, não existe nenhuma razão para se comportar de maneira diferente no mesmo estado em momentos distintos.

Consequentemente, a ação ótima depende apenas do estado atual, e a política ótima é **estacionária**. Políticas para o caso de horizonte infinito são portanto mais simples que aquelas para o caso de horizonte finito, e lidaremos principalmente com o caso de horizonte infinito neste capítulo. (Veremos mais adiante que, para ambientes parcialmente observáveis, o caso de horizonte infinito não é tão simples.) Observe que "horizonte infinito" não significa necessariamente que todas as sequências de estados são infinitas; significa apenas que não existe nenhum prazo final fixo. Em particular, pode haver sequências de estados finitos em um MDP de horizonte infinito contendo um estado terminal.

A próxima pergunta a que devemos responder é como calcular a utilidade de sequências de estados. Na terminologia da teoria de utilidade de multiatributo, cada estado s_i pode ser visto como um atributo da sequência de estado $[s_0, s_1, s_2, \ldots]$. Para obter uma expressão simples em termos dos atributos, precisaremos fazer algum tipo de suposição de independência de preferência. A suposição mais natural é que as preferências do agente entre sequências de estados são **estacionárias**. O caráter estacionário para preferências significa que, se duas sequências de estados $[s_0, s_1, s_2, \ldots]$ e $[s'_0, s'_1, s'_2, \ldots]$ começam com o mesmo estado (isto é, $s_0 = s'_0$), então as duas sequências devem ser ordenadas por preferência, do mesmo modo que as sequências $[s_1, s_2, \ldots]$ e $[s'_1, s'2, \ldots]$. Em linguagem comum, isso significa que, se preferir um futuro a outro que comece amanhã, você ainda deverá preferir esse futuro se ele tiver de começar hoje em vez de amanhã. O caráter estacionário é uma hipótese de aparência bastante inócua mas com consequências muito fortes: sob esse caráter estacionário, existem apenas duas maneiras de atribuir utilidades a sequências:

1. **Recompensas aditivas:** A utilidade de uma sequência de estados é:

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots.$$

O mundo de 4 × 3 da Figura 17.1 utiliza recompensas aditivas. Note que a aditividade foi empregada implicitamente em nosso uso de funções de custo de caminho em algoritmos de busca heurística (Capítulo 3).

2. **Recompensas descontadas:** A utilidade de uma sequência de estados é:

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots,$$

onde o **fator de desconto** γ é um número entre 0 e 1. O fator de desconto descreve a preferência de um agente por recompensas atuais sobre recompensas futuras. Quando γ é próximo de 0, as recompensas no futuro distante são vistas como insignificantes. Quando γ é 1, recompensas descontadas são exatamente equivalentes a recompensas aditivas e, assim, as recompensas aditivas constituem um caso especial de recompensas descontadas. O desconto parece ser um bom modelo de preferências, tanto de animais quanto de humanos, ao longo do tempo. Um fator de desconto γ é equivalente a uma taxa de juros de $(1/\gamma) - 1$.

Por motivos que em breve ficarão claros, vamos assumir recompensas descontadas no restante do capítulo, embora às vezes seja permitido $\gamma = 1$.

Há uma questão que surge de nossa escolha de horizontes infinitos: se o ambiente não contém um estado terminal ou se o agente nunca alcança um desses estados, todos os históricos de ambientes serão infinitamente longos, e as utilidades com recompensas aditivas não descontadas em geral serão infinitas. Agora, podemos concordar que $+\infty$ é melhor que $-\infty$, mas comparar duas sequências de estados, ambas com utilidade $+\infty$, é mais difícil. Existem três soluções, duas das quais já estudamos:

1. Com recompensas descontadas, a utilidade de uma sequência infinita é *finita*. De fato, se $\gamma < 1$ e as recompensas são limitadas por $\pm R_{\max}$, temos:

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max}/(1-\gamma)\,, \tag{17.1}$$

usando a fórmula-padrão para a soma de uma série geométrica infinita.

2. Se o ambiente contém estados terminais *e se o agente oferece a garantia de eventualmente chegar a um deles*, nunca precisaremos comparar sequências infinitas. Uma política que oferece a garantia de alcançar um estado terminal é chamada de **política própria**. Com políticas próprias, podemos usar $\gamma = 1$ (isto é, recompensas aditivas). As três primeiras políticas mostradas na Figura 17.2(b) são próprias, mas a quarta é imprópria. Ela ganha recompensa total infinita ficando afastada dos estados terminais quando a recompensa para os estados não terminais é positiva. A existência de políticas impróprias pode fazer os algoritmos clássicos para resolução de MDPs falharem com recompensas aditivas e, assim, oferece uma boa razão para utilização de recompensas descontadas.
3. As sequências infinitas podem ser comparadas em termos da **recompensa média** obtida por passo de tempo. Suponha que o quadrado (1,1) no mundo 4×3 tenha recompensa de 0,1, enquanto os outros estados não terminais têm recompensa de 0,01. Então, uma política que fizer o melhor possível para ficar em (1,1) terá recompensa média mais alta que uma política que permanecer em outro lugar. A recompensa média é um critério útil para alguns problemas, mas a análise de algoritmos de recompensa média está além do escopo deste livro.

Em suma, o uso de recompensas descontadas apresenta um número menor de dificuldades na avaliação de sequências de estados.

17.1.2 As políticas ótimas e as utilidades dos estados

Ao decidir que a utilidade de uma determinada sequência de estados é a soma das recompensas descontadas obtidas durante a sequência, podemos comparar políticas comparando as utilidades *esperadas* obtidas quando as executamos. Assumimos que o agente está em algum estado inicial s e definimos S_t (uma variável aleatória) como o estado que o agente alcançou no tempo t ao executar uma política determinada π. (Obviamente, $S_0 = s$, o estado em que o agente está agora.) A distribuição de probabilidade sobre as sequências de estados $S_1, S_2, \ldots$, é determinada pelo estado inicial s, a política π, e o modelo de transição do ambiente.

A utilidade esperada obtida executando π a partir de *s* é dada por

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right] , \qquad (17.2)$$

onde a expectativa está relacionada à distribuição de probabilidade sobre sequências de estados determinadas por *s* e π. Agora, entre todas as políticas que o agente poderia escolher executar a partir de *s*, uma (ou mais) terá utilidades esperadas maiores do que todas as outras. Vamos usar π^*_s para indicar uma dessas políticas:

$$\pi^*_s = \operatorname*{argmax}_{\pi} U^{\pi}(s) . \qquad (17.3)$$

Lembre-se que π^*_s é uma política, por isso recomenda uma ação para cada estado; em particular, sua conexão com *s* é ser uma política ótima quando *s* for o estado inicial. Uma consequência notável do uso de utilidades descontadas com horizontes infinitos é que a política ótima é *independente* do estado inicial (é claro, a *sequência de ação* não será independente; lembre-se de que uma política é uma função especificando uma ação para cada estado). Esse fato parece intuitivamente óbvio: se a política π^*_a é ótima começando em *a* e a política π^*_b é ótima começando em *b*; então, quando alcançarem um terceiro estado *c*, não há nenhuma boa razão para que discordem uma da outra, ou de π^*_c, sobre o que fazer a seguir.[2] Assim, podemos simplesmente escrever π* para uma política ótima.

Dada essa definição, a utilidade verdadeira de um estado é simplesmente $U^{\pi^*}(s)$, isto é, a soma esperada de recompensas descontadas se o agente executa uma política ótima. Escrevemos como *U*(*s*), de acordo com a notação usada no Capítulo 16 para a utilidade de um resultado. Note que *U*(*s*) e *R*(*s*) são quantidades bastante diferentes; *R*(*s*) é a recompensa "a curto prazo" por estar em *s*, enquanto *U*(*s*) é a recompensa total "a longo prazo" de *s* em diante. A Figura 17.3 mostra as utilidades para o mundo 4 × 3. Note que as utilidades são mais altas para estados mais próximos à saída +1, porque são necessários menos passos para alcançar a saída.

3	0,812	0,868	0,918	+1
2	0,762		0,660	–1
1	0,705	0,655	0,611	0,388
	1	2	3	4

Figura 17.3 As utilidades dos estados no mundo 4 × 3, calculadas com γ = 1 e *R*(*s*) = –0,04 para estados não terminais.

A função utilidade *U*(*s*) permite ao agente selecionar ações usando o princípio de utilidade esperada máxima do Capítulo 16, isto é, escolher a ação que maximiza a utilidade esperada do estado subsequente:

$$\pi^*(s) = \underset{a \in A(s)}{\text{argmax}} \sum_{s'} P(s' \mid s, a) U(s') . \tag{17.4}$$

Nas duas próximas seções descreveremos algoritmos para encontrar políticas ótimas.

17.2 ITERAÇÃO DE VALOR

Nesta seção, apresentaremos um algoritmo, chamado de **iteração de valor**, para calcular uma política ótima. A ideia básica é calcular a utilidade de cada estado e, em seguida, usar as utilidades do estado para selecionar uma ação ótima em cada estado.

17.2.1 A equação de Bellman para utilidades

A Seção 17.1.2 definiu a utilidade de estar em um estado como a soma esperada de recompensas descontadas a partir desse momento. A partir disso, segue que há uma relação direta entre a utilidade de um estado e a utilidade de seus vizinhos: *a utilidade de um estado é a recompensa imediata correspondente a esse estado mais a utilidade descontada esperada do próximo estado, assumindo que o agente escolha a ação ótima.* Isto é, a utilidade de um estado s é dada por

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s') . \tag{17.5}$$

Essa equação é chamada de **equação de Bellman**, em homenagem a Richard Bellman (1957). As utilidades dos estados — definidas pela Equação 17.2, como a utilidade esperada das sequências de estados subsequentes — são soluções do conjunto das equações de Bellman. Na verdade, são as *únicas* soluções, como mostramos na Seção 17.2.3.

Vamos examinar uma das equações de Bellman para o mundo 4 × 3. A equação para o estado (1,1) é:

$$\begin{aligned} U(1,1) = -0.04 + \gamma \max[&0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && (Acima)\\ &0.9U(1,1) + 0.1U(1,2), && (Esquerda)\\ &0.9U(1,1) + 0.1U(2,1), && (Abaixo)\\ &0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)]. && (Direita) \end{aligned}$$

Quando inserirmos os números da Figura 17.3, descobriremos que *Acima* é a melhor ação.

17.2.2 O algoritmo de iteração de valor

A equação de Bellman é a base do algoritmo de iteração de valor para resolução de MDPs. Se houver n estados possíveis, haverá n equações de Bellman, uma para cada estado. As n equações

contêm n incógnitas — as utilidades dos estados. Então, gostaríamos de resolver essas equações simultâneas para encontrar as utilidades. Porém, existe um problema: as equações são *não lineares* porque o operador "max" não é um operador linear. Enquanto sistemas de equações lineares podem ser resolvidos com rapidez usando-se técnicas de álgebra linear, sistemas de equações não lineares são mais problemáticos. Podemos experimentar uma abordagem *iterativa*. Começaremos com valores iniciais arbitrários para as utilidades, calculamos o lado direito da equação e o inserimos no lado esquerdo, atualizando assim a utilidade de cada estado a partir das utilidades de seus vizinhos. Repetimos esse processo até chegarmos a um equilíbrio. Seja $U_i(s)$ o valor de utilidade para o estado s na i-ésima iteração. A etapa de iteração, chamada **atualização de Bellman**, é feita da seguinte forma:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s'), \tag{17.6}$$

onde se assume que a atualização será aplicada simultaneamente a todos os estados em cada iteração. Se aplicarmos a atualização de Bellman com frequência infinita, teremos a garantia de alcançar um equilíbrio (consulte a Seção 17.2.3) e, nesse caso, os valores finais de utilidade deverão ser soluções para as equações de Bellman. De fato, eles também são as *únicas* soluções, e a política correspondente (obtida com o uso da Equação 17.4) é ótima. O algoritmo, chamado ITERAÇÃO-DE-VALOR, é mostrado na Figura 17.4.

```
função ITERAÇÃO-DE-VALOR(mdp, ε) retorna uma função utilidade
   entradas: mdp, um MDP com estados S, ações A(s), modelo de transição P(s'|s,a), recompensas R(s), desconto γ,
             ε, o erro máximo permitido na utilidade de qualquer estado
   variáveis locais: U, U', vetores de utilidades para estados em S, inicialmente zero
                     δ, a mudança máxima na utilidade de qualquer estado em uma iteração

   repita
      U ← U'; δ ← 0
      para cada estado s em S faça
         U'[s] ← R[s] + γ max_{a ∈ A(s)} Σ_{s'} P(s' | s, a) U[s']
         se |U'[s] − U[s]| > δ então δ ← |U'[s] − U[s]|
   até δ < ε(1 − γ)/γ
   retornar U
```

Figura 17.4 O algoritmo de iteração de valor para calcular utilidades de estados. A condição de término vem da Equação 17.8.

Podemos aplicar a iteração de valor ao mundo 4 × 3 da Figura 17.1(a). Começando com valores iniciais iguais a zero, as utilidades evoluem como mostra a Figura 17.5(a). Note como os estados a diferentes distâncias de (4,3) acumulam recompensa negativa até um caminho para (4,3) ser encontrado; daí em diante, as utilidades começam a aumentar. Podemos imaginar o algoritmo de iteração de valor como a *propagação de informações* pelo espaço de estados por meio de atualizações locais.

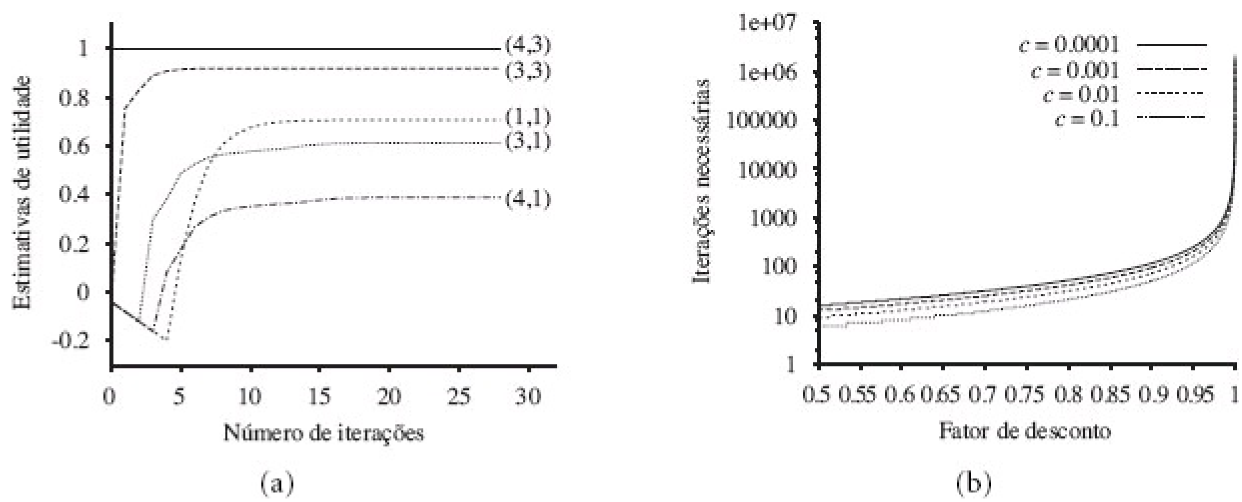


Figura 17.5 (a) Grafo mostrando a evolução das utilidades de estados selecionados usando iteração de valor. (b) O número de iterações de valor *k* necessárias para garantir um erro no máximo igual a $\varepsilon = c \cdot R_{\max}$, para diferentes valores de *c*, como uma função do fator de desconto γ.

17.2.3 Convergência da iteração de valor

Dissemos que a iteração de valor eventualmente converge para um único conjunto de soluções das equações de Bellman. Nesta seção, explicaremos por que isso acontece. Introduziremos algumas ideias matemáticas úteis ao longo do processo e obteremos alguns métodos para avaliar o erro na função utilidade devolvida quando o algoritmo é terminado prematuramente; isso é útil porque significa que não teremos de continuar para sempre. Essa seção é bastante técnica.

O conceito básico usado para mostrar que a iteração de valor converge é a noção de **contração**. A *grosso modo*, uma contração é uma função de um único argumento que, ao ser aplicada a duas entradas diferentes, uma de cada vez, produz dois valores de saída que estão "mais próximos entre si" por pelo menos um fator constante, em relação às entradas originais. Por exemplo, a função "dividir por dois" é uma contração porque, depois de dividirmos dois números quaisquer por dois, sua diferença é reduzida à metade. Note que a função "dividir por dois" tem um ponto fixo, isto é, zero, que não é alterado pela aplicação da função. A partir desse exemplo, podemos distinguir duas propriedades importantes de contrações:

- Uma contração tem apenas um ponto fixo; se houvesse dois pontos fixos, eles não ficariam mais próximos um do outro quando a função fosse aplicada e não seria uma contração.
- Quando a função é aplicada a qualquer argumento, o valor deve ficar mais próximo do ponto fixo (porque o ponto fixo não se move) e, assim, a aplicação repetida de uma contração sempre alcança o ponto fixo no limite.

Agora, vamos supor que visualizamos a atualização de Bellman (Equação 17.6) como um operador *B* que é aplicado simultaneamente para atualizar a utilidade de todo estado. Seja U_i o vetor de utilidades para todos os estados na *i*-ésima iteração. Então, a equação da atualização de Bellman pode ser escrita como

$U_{i+1} \leftarrow BU_i$.

Em seguida, precisamos de um modo de medir distâncias entre vetores de utilidade. Utilizaremos a **norma max**, que mede o "comprimento" de um vetor pelo valor absoluto de seu maior componente:

$$||U|| = \max_s |U(s)| .$$

Com essa definição, a "distância" entre dois vetores, $\|U - U'\|$, é a diferença máxima entre dois elementos correspondentes quaisquer. O principal resultado desta seção é: *sejam* U_i e U'_i *dois vetores de utilidade quaisquer.Então, temos*:

$$||BU_i - BU'_i|| \leq \gamma \, ||U_i - U'_i||. \quad (17.7)$$

Isto é, a atualização de Bellman é uma contração por um fator γ *no espaço de vetores de utilidade* (o Exercício 17.6 fornece alguma orientação para provar esta afirmação). Assim, a partir das propriedades de contrações em geral, segue que a iteração de valor sempre converge para uma solução única das equações de Bellman, sempre que $\gamma < 1$.

Podemos também utilizar a propriedade de contração para analisar *a taxa* de convergência para uma solução. Em particular, podemos substituir U'_i na Equação (17.7) pelas utilidades *verdadeiras* U, para as quais $BU = U$. Então, obtemos a desigualdade:

$$||BU_i - U|| \leq \gamma \, ||U_i - U||.$$

Assim, se visualizarmos $\|U_i - U\|$ como o *erro* na estimativa U_i, veremos que o erro é reduzido por um fator de pelo menos γ em cada iteração. Isso significa que a iteração de valor converge de forma exponencialmente rápida. Podemos calcular o número de iterações necessárias para alcançar um limite de erro especificado ε, da seguinte forma: primeiro, vimos na Equação 17.1 que as utilidades de todos os estados são limitadas por $\pm R_{max}/(1 - \gamma)$. Isso significa que o erro inicial máximo $\|U_0 - U\| \leq 2R_{max}/(1 - \gamma)$. Vamos supor que sejam realizadas N iterações para alcançar um erro de no máximo ε. Então, como o erro é reduzido por pelo menos γ em cada vez, é necessário que ($\gamma^N \cdot 2R_{max}/(1 - \gamma) \leq \varepsilon$. Usando logaritmos, descobrimos que

$$N = \lceil \log(2R_{max}/\epsilon(1 - \gamma))/\log(1/\gamma) \rceil$$

iterações bastam. A Figura 17.5(b) mostra como N varia com γ para diferentes valores da razão ε/R_{max}. A boa notícia é que, devido à convergência exponencialmente rápida, N não depende muito da razão ε/R_{max}. A má notícia é que N cresce rapidamente à medida que γ se aproxima de 1. Podemos obter a convergência rápida se tornarmos γ pequeno, mas isso efetivamente dá ao agente um horizonte curto e pode anular os efeitos a longo prazo das ações do agente.

O limite de erro no parágrafo anterior nos dá alguma ideia dos fatores que influenciam o tempo de execução do algoritmo, mas às vezes é excessivamente conservador como um método para decidir quando interromper a iteração. Para este último propósito, podemos utilizar um limite relacionando o erro ao tamanho da atualização de Bellman em qualquer iteração dada. A partir da propriedade de

contração (Equação 17.7), podemos mostrar que, se a atualização for pequena (isto é, se a utilidade não mudar muito para nenhum estado), então o erro, comparado à função utilidade verdadeira, também será pequeno. Mais precisamente,

$$\text{se} \quad ||U_{i+1} - U_i|| < \epsilon(1-\gamma)/\gamma, \quad \text{então} \quad ||U_{i+1} - U|| < \epsilon. \tag{17.8}$$

Essa é a condição de término usada no algoritmo ITERAÇÃO-DE-VALOR da Figura 17.4.

Até agora, analisamos o erro na função utilidade devolvida pelo algoritmo de iteração de valor. *Porém, o que realmente importa para o agente é como ele se sairá se tomar suas decisões com base em sua função utilidade*. Vamos supor que, depois de i iterações da iteração de valor, o agente tenha uma estimativa U_i da utilidade verdadeira U e obtenha a política π_i de UME π_i com base na observação para frente de um passo usando U_i (como na Equação 17.4). O comportamento resultante será quase tão bom quanto o comportamento ótimo? Essa é uma questão crucial para qualquer agente real, e sua resposta é sim. $U^{\pi_i}(s)$ é a utilidade obtida se π_i é executada a partir de s, e a **perda de política** $\|U^{\pi_i} - U\|$ é o máximo que o agente pode perder executando π_i em lugar da política ótima π^*. A perda de política de π_i está relacionada ao erro em U_i pela desigualdade a seguir:

$$\text{se} \quad ||U_i - U|| < \epsilon, \quad \text{então} \quad ||U^{\pi_i} - U|| < 2\epsilon\gamma/(1-\gamma). \tag{17.9}$$

Na prática, com frequência ocorre que π_i se torna ótima bem antes de U_i ter convergido. A Figura 17.6 mostra como o erro máximo em U_i e a perda de política se aproximam de zero à medida que o processo de iteração de valor prossegue para o ambiente 4 × 3 com $\gamma = 0{,}9$. A política π_i é ótima quando $i = 4$, embora o erro máximo em U_i ainda seja 0,46.

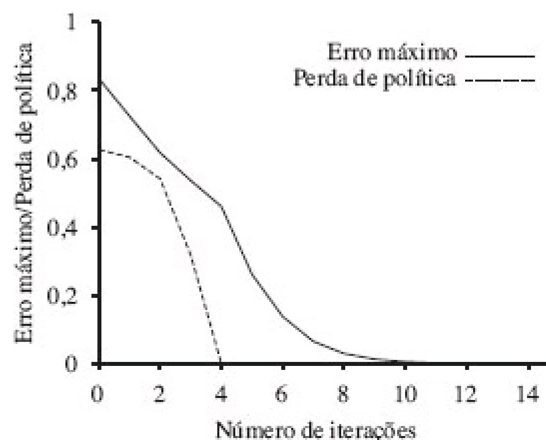


Figura 17.6 O erro máximo $\|U_i - U\|$ das estimativas de utilidade e a perda de política $\|U^{\pi'} - U\|$, como uma função do número de iterações da iteração de valor.

Agora temos tudo o que precisamos para utilizar a iteração de valor na prática. Sabemos que ela converge para as utilidades corretas, podemos limitar o erro nas estimativas de utilidade se pararmos após um número finito de iterações e podemos limitar a perda de política que resulta da execução da política UME correspondente. Como observação final, todos os resultados desta seção dependem do desconto com $\gamma < 1$. Se $\gamma = 1$ e o ambiente contiver estados terminais, um conjunto semelhante de resultados de convergência e de limites de erro poderá ser derivado sempre que certas condições

técnicas forem satisfeitas.

17.3 ITERAÇÃO DE POLÍTICA

Na seção anterior, observamos que é possível conseguir uma política ótima até mesmo quando a estimativa de função utilidade é inexata. Se uma ação é claramente melhor que todas as outras, a magnitude exata das utilidades nos estados envolvidos não precisa ser exata. Essa ideia sugere um caminho alternativo para encontrar políticas ótimas. O algoritmo de **iteração de política** alterna as duas etapas a seguir, começando com alguma política inicial π_0:

- **Avaliação de política:** Dada uma política π_i, calcular $U_i = U^{\pi_i}$, a utilidade de cada estado se π_i fosse executada.
- **Aperfeiçoamento de política:** Calcular uma nova política UME π_{i+1} utilizando a observação para frente de um passo baseada em U_i (como na Equação (17.4).

O algoritmo termina quando a etapa de aperfeiçoamento da política não produz nenhuma mudança nas utilidades. Nesse ponto, sabemos que a função utilidade U_i é um ponto fixo da atualização de Bellman, e portanto ela é uma solução para as equações de Bellman, e π_i deve ser uma política ótima. Como existe apenas um número finito de políticas para um espaço de estados finito e podemos mostrar que cada iteração produz uma política melhor, a iteração de políticas tem de terminar. O algoritmo é mostrado na Figura 17.7.

função ITERAÇÃO-DE-POLÍTICA(*mdp*) **retorna** uma política
  entradas: *mdp*, um MDP com estados S, ações $A(s)$, modelo de transição $P(s' \mid s, a)$
  variáveis locais: U, um vetor de utilidades para estados em S, inicialmente zero
    π, um vetor de política indexado pelo estado, inicialmente aleatório
  repita
    $U \leftarrow$ AVALIAÇÃO-DE-POLÍTICA(π, U, *mdp*)
    inalterado? ← verdadeiro
    para cada estado s **em** S **faça**
    se $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **então faça**
      $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
      inalterado? ← falso
  até *inalterado?*
  retornar π

Figura 17.7 Algoritmo de iteração de política para calcular de uma política ótima.

A etapa de aperfeiçoamento da política é obviamente direta; porém, como a rotina AVALIAÇÃO-DE-POLÍTICA pode ser implementada? Na realidade, fazer isso é muito mais simples que resolver as equações de Bellman (o que é feito pela iteração de valor) porque a ação em cada estado é fixada pela política. Na i-ésima iteração, a política π_i especifica a ação $\pi_i(s)$ no estado s. Isso significa que temos uma versão simplificada da equação de Bellman (17.5) relacionando a utilidade de s (sob π_i)

às utilidades de seus vizinhos:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s') . \quad (17.10)$$

Por exemplo, vamos supor que π_i seja a política mostrada na Figura 17.2(a). Então, temos $\pi_i(1,1)$ = *Acima*, $\pi_i(1, 2)$ = *Acima*, e assim por diante, e as equações de Bellman simplificadas são:

$$U_i(1,1) = -0{,}04 + 0{,}8U_i(1,2) + 0{,}1U_i(1,1) + 0{,}1U_i(2,1) ,$$
$$U_i(1,2) = -0{,}04 + 0{,}8U_i(1,3) + 0{,}2U_i(1,2).$$
$$\vdots$$

O ponto importante é que essas equações são *lineares* porque o operador "max" foi removido. Para n estados, temos n equações lineares com n incógnitas, que podem ser resolvidas exatamente no tempo $O(n^3)$ por métodos clássicos de álgebra linear.

Para espaços de estados pequenos, a avaliação de política usando métodos de solução exata em geral é a abordagem mais eficiente. Para os espaços de estados grandes, o tempo $O(n^3)$ talvez seja proibitivo. Felizmente, não é necessário fazer a avaliação de política *exata*. Em vez disso, podemos executar algum número de passos de iteração de valor simplificada (simplificada porque a política é fixa) para fornecer uma aproximação razoavelmente boa das utilidades. A atualização de Bellman simplificada para esse processo é

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s') ,$$

e é repetida k vezes para produzir a próxima estimativa de utilidade. O algoritmo resultante é chamado **iteração de política modificada**. Com frequência, ele é muito mais eficiente que a iteração de política clássica ou a iteração de valor.

Os algoritmos que descrevemos até agora exigem atualização da utilidade ou da política para todos os estados de uma vez. Ocorre que isso não é estritamente necessário. De fato, em cada iteração, podemos escolher *qualquer subconjunto* de estados e aplicar um dos tipos de atualização (aperfeiçoamento de política ou *uma* iteração de valor simplificada) a esse subconjunto. Esse algoritmo bem geral é chamado **iteração de política assíncrona**. Dadas certas condições sobre a política inicial e sobre a função utilidade, a iteração de política assíncrona oferece a garantia de convergir para uma política ótima. A liberdade de escolher quaisquer estados para trabalhar significa que podemos projetar algoritmos heurísticos muito mais eficientes — por exemplo, algoritmos que se concentram em atualizar os valores de estados que provavelmente serão alcançados por uma boa política. Isso faz muito sentido na vida real: se uma pessoa não tem nenhuma intenção de se lançar em um precipício, não devemos perder tempo nos preocupando com o valor exato dos estados resultantes.

17.4 MDPS PARCIALMENTE OBSERVÁVEIS

A descrição de processos de decisão de Markov na Seção 17.1 pressupôs que o ambiente era

completamente observável. Com essa suposição, o agente sempre sabe em que estado se encontra. Isso, combinado com a hipótese de Markov para o modelo de transição, significa que a política ótima depende unicamente do estado atual. Quando o ambiente é apenas **parcialmente observável**, a situação é muito menos clara. O agente não sabe necessariamente em que estado se encontra e, portanto, não pode executar a ação $\pi(s)$ recomendada para esse estado. Além disso, a utilidade de um estado s e a ação ótima em s não dependem apenas de s, mas também *do quanto o agente sabe* quando está em s. Por essas razões, os **MDPs parcialmente observáveis** (ou POMDPs) em geral são considerados muito mais difíceis que os MDPs comuns. No entanto, não podemos evitar POMDPs porque o mundo real é um deles.

17.4.1 Definição de POMDPs

Para compreender os POMDPs, primeiro devemos defini-los corretamente. Um POMDP tem os mesmos elementos que um MDP — o modelo de transição $P(s' | s, a)$, as ações $A(s)$ e a função recompensa $R(s)$ —, mas, como os problemas de busca parcialmente observáveis da Seção 4.4, ele também tem um **modelo de sensoriamento** $P(e \mid s)$. Aqui, como no Capítulo 15, o modelo de sensoriamento especifica a probabilidade de perceber a evidência e no estado s.[3] Por exemplo, podemos converter o mundo 4 × 3 da Figura 17.1 em um POMDPs pela adição de um sensoriamento ruidoso ou parcial em vez de assumir que o agente conhece a sua localização exata. Tal sensoriamento pode medir o *número de paredes adjacentes*, que vem a ser 2 em todos os quadrados não terminais, exceto os da terceira coluna, onde o valor é 1; uma versão ruidosa pode dar o valor errado com probabilidade 0,1.

Nos Capítulos 4 e 11, estudamos problemas de planejamento não determinístico e parcialmente observável, e identificamos o **estado de crença** — o conjunto de estados reais em que o agente poderia estar — como um conceito fundamental para descrever e calcular soluções. Em POMDPs, o estado de crença b *torna*-se uma *distribuição de probabilidade* sobre todos os estados possíveis, como no Capítulo 15. Por exemplo, o estado de crença inicial para o POMDP 4 × 3 poderia ter a distribuição uniforme sobre nove estados não terminais, ou seja, $\langle \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0, 0 \rangle$. Escreveremos $b(s)$ para representar a probabilidade atribuída ao estado real s pelo estado de crença b. O agente pode calcular seu estado de crença atual como a distribuição de probabilidade condicional sobre os estados reais, dada a sequência de observações e ações até o momento. Em essência, essa é a tarefa de **filtragem** descrita no Capítulo 15. A equação de filtragem recursiva básica (15.5) mostra como calcular o novo estado de crença a partir do estado de crença anterior e a nova evidência. No caso de POMDPs, também temos uma ação a considerar, mas o resultado é essencialmente o mesmo. Se $b(s)$ era o estado de crença anterior, e o agente executa a ação a e percebe a evidência e, então o novo estado de crença é dado por

$$b'(s') = \alpha\, P(e \mid s') \sum_{s} P(s' \mid s, a) b(s) \,,$$

onde é uma constante de normalização que torna a soma do estado de crença igual a 1. Por analogia com o operador de atualização de filtragem (Seção 15.2.1), podemos escrever como

$$b' = \text{PARAFRENTE}(b, a, e). \quad (17.11)$$

No POMDP 4 × 3 suponha que o agente se mova para a *Esquerda* e seu sensor relate uma parede adjacente; então, é bem provável (embora não garantido, porque tanto o movimento como o sensor são ruidosos) que o agente agora esteja em (3,1). O Exercício 17.13 pede para calcular os valores de probabilidade exata para o novo estado de crença.

A ideia fundamental necessária para entender os POMDPs é: *a ação ótima depende apenas do estado de crença atual do agente*. Isto é, a política ótima pode ser descrita por um mapeamento $\pi^*(b)$ de estados de crença para ações. Ela *não* depende do estado *real* em que o agente se encontra. Isso é bom, porque o agente não conhece seu estado real; tudo o que ele conhece é o estado de crença. Consequentemente, o ciclo de decisão de um agente POMDP pode ser quebrado nos três seguintes passos:

1. Dado o estado de crença atual b, executar a ação $a = \pi^*(b)$.
2. Receber a percepção e.
3. Definir o estado de crença atual como PARAFRENTE(b, a, e) e repetir.

Agora, podemos pensar que os POMDPs precisam fazer uma busca no espaço de estados de crença, exatamente como os métodos para problemas sem sensores e de contingência do Capítulo 4. A principal diferença é que o espaço de estados de crença de POMDPs é *contínuo* porque um estado de crença de um POMDP é uma distribuição de probabilidade. Por exemplo, um estado de crença para o mundo 4 × 3 é um ponto em um espaço contínuo de 11 dimensões. Uma ação altera o estado de crença, não apenas o estado físico. Assim, a ação é avaliada, pelo menos em parte, de acordo com as informações que o agente adquire como resultado. Portanto, os POMDPs incluem o valor da informação (Seção 16.6) como um componente do problema de decisão.

Vamos examinar mais cuidadosamente o resultado das ações. Em particular, vamos calcular a probabilidade de um agente no estado de crença b alcançar o estado de crença b' depois da execução da ação a. Se tivéssemos conhecimento da ação *e da percepção subsequente*, a Equação 17.11 forneceria uma atualização *determinística* para o estado de crença: b' = PARAFRENTE(b, a, e). É claro que a percepção subsequente ainda não é conhecida e, assim, o agente pode chegar a um dos vários estados de crença b' possíveis, dependendo da percepção que ocorre. A probabilidade de perceber e, dado que a foi executada a partir do estado de crença b, é dada pelo somatório sobre todos os estados reais s', que o agente poderia alcançar:

$$\begin{aligned} P(e|a,b) &= \sum_{s'} P(e|a,s',b)P(s'|a,b) \\ &= \sum_{s'} P(e\,|\,s')P(s'|a,b) \\ &= \sum_{s'} P(e\,|\,s')\sum_{s} P(s'\,|\,s,a)b(s)\,. \end{aligned}$$

Vamos escrever a probabilidade de alcançar b' a partir de b, dada a ação a, como $P(b'|\, b, a)$. Então, isso nos dá

$$P(b' \mid b, a) = P(b'|a, b) = \sum_e P(b'|e, a, b)P(e|a, b)$$
$$= \sum_e P(b'|e, a, b) \sum_{s'} P(e \mid s') \sum_s P(s' \mid s, a)b(s) , \qquad (17.12)$$

onde $P(b' \mid e, a, b)$ é 1 se b' = PARAFRENTE(b, a, e) e 0 em caso contrário.

A Equação 17.12 pode ser vista como a definição de um modelo de transição para o espaço de estados de crença. Também podemos definir uma função de recompensa para estados de crença (isto é, a recompensa esperada para os estados reais em que o agente poderia estar):

$$\rho(b) = \sum_s b(s)R(s) .$$

Juntos, $P(b' \mid b,a)$ e $P(b)$ definem um MDP *observável* sobre o espaço de estados de crença. Além disso, é possível mostrar que uma política ótima para esse MDP, $\pi^*(b)$, também é uma política ótima para o POMDP original. Em outras palavras, *a resolução de um POMDP em um espaço de estados físicos pode ser reduzida à resolução de um MDP no espaço de estados de crença correspondente*. Esse fato talvez seja menos surpreendente se lembrarmos que, por definição, o estado de crença é sempre observável para o agente.

Note que, embora tenhamos reduzido os POMDPs a MDPs, o MDP que obtemos tem um espaço de estados contínuo (e, em geral, com número elevado de dimensões). Nenhum dos algoritmos de MDP descritos nas Seções 17.2 e 17.3 se aplica diretamente a tais MDPs. As próximas duas subseções descrevem um algoritmo de iteração de valor projetado especificamente para POMDPs e um algoritmo de tomada de decisão on-line, semelhante ao desenvolvido para jogos no Capítulo 5.

17.4.2 Iteração de valor para POMDPs

A Seção 17.2 descreveu um algoritmo de iteração de valor que calcula um valor de utilidade para cada estado. Com os estados de crença infinitos, precisamos ser mais criativos. Considere uma política ótima π^* e sua aplicação em um estado de crença específico b: a política gera uma ação; então, para cada percepção subsequente, o estado de crença é atualizado e uma nova ação é gerada, e assim por diante. Para esse b específico, portanto, a política é exatamente equivalente a um **plano condicional**, conforme definido no Capítulo 4 para problemas não determinísticos e parcialmente observáveis. Em vez de pensar sobre as políticas, vamos pensar sobre planos condicionais e como a utilidade esperada da execução de um plano condicional fixo varia com o estado de crença inicial. Faremos duas observações:

1. Seja $\alpha_p(s)$ a utilidade de execução de um plano condicional fixo p que inicia em um estado físico s. Então, a utilidade esperada da execução de p no estado de crença b é exatamente $\sum_s b(s)\alpha_p(s)$ ou $b \cdot \alpha_p$ se considerarmos ambos como vetores. Assim, a utilidade esperada de um plano condicional fixo vai variar *linearmente* com b, ou seja, corresponde a um hiperplano no espaço de crença.

2. Em qualquer estado de crença b dado a política ótima, vai escolher executar o plano condicional com a maior utilidade esperada, e a utilidade esperada de b sob política ótima será exatamente a utilidade do plano condicional:

$$U(b) = U^{\pi^*}(b) = \max_p b \cdot \alpha_p .$$

Se a política ótima π* escolher executar p a partir de b, então é razoável esperar que possamos escolher executar p nos estados de crença que estão muito perto de b. Na verdade, se limitarmos a profundidade dos planos condicionais, haverá apenas um número finito de tais planos e o espaço contínuo de estados de crença geralmente será dividido em *regiões*, cada uma correspondendo a determinado plano condicional que é o ótimo daquela região.

Dessas duas observações, vemos que a função utilidade $U(b)$ nos estados de crença, sendo o máximo de um conjunto de hiperplanos, será *linear por partes* e *convexa*.

Para ilustrar, utilizaremos um mundo simples de dois estados. Os estados são rotulados como 0 e 1, com $R(0) = 0$ e $R(1) = 1$. Há duas ações: *Permanecer*, faz com que o agente fique no mesmo estado com probabilidade 0,9, e *Ir,* faz com que o agente mova para o outro estado com probabilidade 0,9. Por enquanto vamos assumir o fator de desconto $\gamma = 1$. O sensor informa o estado correto com probabilidade 0,6. Obviamente, o agente deve *Permanecer* quando acredita que está no estado 1 e *Ir* quando acredita que está no estado 0.

A vantagem de um mundo de dois estados é que o espaço de crença pode ser visto como unidimensional porque as duas probabilidades devem somar 1. Na Figura 17.8 (a), o eixo x representa o estado de crença, definido por $b(1)$, a probabilidade de estar no estado 1. Vamos considerar os planos de um passo [*Permanecer*] e [*Ir*], cada um dos quais recebe a recompensa pelo estado corrente seguido pela recompensa (descontada) para o estado alcançado após a ação:

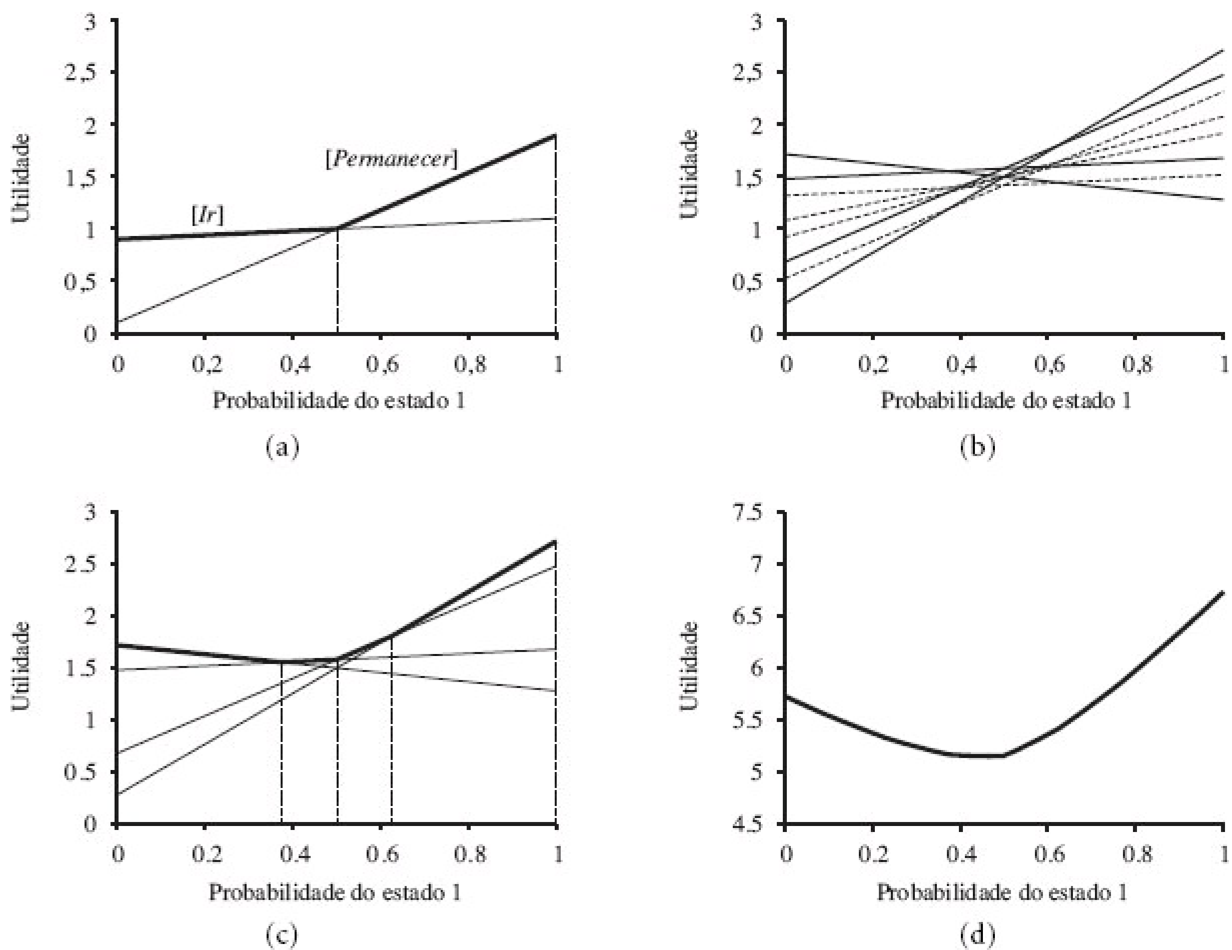


Figura 17.8 (a) Utilidade de dois planos de um passo como função do estado de crença inicial $b(1)$ para o mundo de dois estados, com a função utilidade correspondente mostrada em negrito. (b) Utilidades para oito planos distintos de dois passos. (c) Utilidades para quatro planos de dois passos não dominados. (d) Função utilidade para planos ótimos de oito passos.

$$\alpha_{[Permanecer]}(0) = R(0) + \gamma(0{,}9R(0) + 0{,}1R(1)) = 0{,}1$$
$$\alpha_{[Permanecer]}(1) = R(1) + \gamma(0{,}9R(1) + 0{,}1R(0)) = 1{,}9$$
$$\alpha_{[Ir]}(0) = R(0) + \gamma(0{,}9R(1) + 0{,}1R(0)) = 0{,}9$$
$$\alpha_{[Ir]}(1) = R(1) + \gamma(0{,}9R(0) + 0{,}1R(1)) = 1{,}1$$

Os hiperplanos (linhas, nesse caso) para $b \cdot \alpha_{[Permanecer]}$ e $b \cdot \alpha_{[Ir]}$ são mostrados na Figura 17.8(a) e o seu máximo é mostrado em negrito. A linha em negrito representa, portanto, a função utilidade para o problema de horizonte finito que permite apenas uma ação, e, em cada "parte" da função utilidade linear por partes, a ação ótima é a primeira ação do plano condicional correspondente. Nesse caso, a política de um passo ótima é *Permanecer* quando $b(1) > 0{,}5$ e *Ir,* caso contrário.

Uma vez que temos utilidades $\alpha_p(s)$ para todos os planos condicionais p de profundidade 1 em cada estado físico s, podemos calcular as utilidades para os planos condicionais de profundidade 2 considerando cada primeira ação possível, cada percepção subsequente possível e, então, cada forma de escolher um plano de profundidade 1 para executar para cada percepção:

[*Permanecer*; **se** *Percepção* = 0 **então** *Permanecer* **senão** *Permanecer*]
[*Permanecer*; **se** *Percepção* = 0 **então** *Permanecer* **senão** *Ir*]...

Ao todo, há oito planos distintos de profundidade 2, e suas utilidades são mostradas na Figura 17.8(b). Note que quatro dos planos mostrados como linhas tracejadas são subótimos em todo o espaço de crença — dizemos que esses planos são **dominados**, e eles não precisam mais ser considerados. Há quatro planos não dominados, cada um dos quais é ótimo em uma região específica, como mostrado na Figura17.8(c). As regiões dividem o espaço de estados de crença.

Repetimos o processo para a profundidade 3, e assim por diante. Em geral, seja p um plano condicional de profundidade d cuja ação inicial é a e cujo subplano de profundidade $d - 1$ para a percepção e é $p.e$; então

$$\alpha_p(s) = R(s) + \gamma \left(\sum_{s'} P(s' \mid s, a) \sum_{e} P(e \mid s') \alpha_{p.e}(s') \right) . \qquad (17.13)$$

Essa recursão naturalmente nos dá um algoritmo de iteração de valor, que está esboçado na Figura 17.9. A estrutura do algoritmo e sua análise de erro são semelhantes aos algoritmos de iteração de valor básico da Figura 17.4; a principal diferença é que, em vez de calcular um número de utilidade para cada estado, o mantém uma coleção de planos não dominados com seus hiperplanos de utilidade. A complexidade do algoritmo depende, principalmente, de quantos planos são gerados. Dadas $|A|$ ações e $|E|$ observações possíveis, é fácil mostrar que existem $|A|^{O(|E|^{d-1})}$ planos distintos com profundidade d. Mesmo para o simples mundo de dois estados, com $d = 8$, o número exato é 2^{255}. A eliminação dos planos dominados é fundamental para reduzir esse crescimento duplamente exponencial: o número de planos não dominados com $d = 8$ é de apenas 144. A função utilidade para esses 144 planos é mostrada na Figura 17.8(d).

funçãoITERAÇÃO-DE-VALOR-POMDP(pomdp, ∈) **retorna** uma função utilidade
entradas: *pomdp*, um POMDP com estados *S*, ações *A*(*s*), modelo de transição *P*(*s*′ | *s*, *a*),
modelo de sensoriamento *P*(*e* | *s*), recompensa *R*(*s*), desconto γ
∈, erro máximo permitido na utilidade de qualquer estado
variáveis locais: *U*, *U*′, conjuntos de planos *p* associados com vetores de utilidade a_p
U′ ← conjunto contendo apenas o plano vazio [ ], com $a_{[\,]}$ (*s*) = *R*(*s*)
repetir
U ← *U*′
U′ ← conjunto de todos os planos que consistem em uma ação e, para cada próxima percepção possível, um plano em *U* com vetores de utilidade calculados de acordo com a Equação 17.13
U′ ← REMOVER-PLANOS-DOMINADOS (*U*′)
até DIFERENÇA-MAX (*U*, *U*′) < ∈(1 − γ)/γ
retornar *U*

Figura 17.9 Esboço de alto nível do algoritmo de iteração de valor para POMDPs. A etapa REMOVER-PLANOS-DOMINADOS e o teste da DIFERENÇA-MAX são implementados normalmente como programas lineares.

Observe que, apesar do estado 0 ter utilidade mais baixa do que o estado 1, os estados de crença intermediários têm utilidade ainda mais baixa porque o agente não tem a informação necessária para escolher uma boa ação. É por isso que a informação tem valor no sentido definido na Seção 16.6, e as políticas ótimas em POMDPs muitas vezes incluem ações de coleta de informações.

Dada tal função utilidade, pode-se extrair uma política executável examinando qual hiperplano é ótimo em qualquer estado de crença dado b e executando a primeira ação do plano correspondente. N a Figura 17.8(d), a política ótima correspondente ainda é a mesma que para os planos de profundidade 1: *Permanecer* quando $b(1) > 0,5$ e *Ir*, caso contrário.

Na prática, o algoritmo de iteração de valor na Figura 17.9 é muito ineficiente para problemas maiores — mesmo o POMDP 4 × 3 é muito difícil. A principal razão é que, dados n planos condicionais no nível d, o algoritmo constrói $|A| \cdot n^{|E|}$ planos condicionais no nível $d + 1$ antes de eliminar os dominados. Desde os anos 1970, quando esse algoritmo foi desenvolvido, houve vários avanços, incluindo formas mais eficientes de iteração de valor e de vários tipos de algoritmos de iteração de política. Alguns deles são discutidos nas notas ao final do capítulo. No entanto, para POMDPs em geral, encontrar políticas ótimas é muito difícil (PSPACE-difícil, na verdade, ou seja, muito difícil mesmo). Problemas com poucas dúzias de estados muitas vezes são inviáveis. A próxima seção descreve um método diferente, aproximado, para resolver POMDPs, baseado em busca de observação para frente.

17.4.3 Agentes on-line de POMDPs

Nesta seção, esboçaremos uma abordagem simples para projeto de agentes destinados a ambientes estocásticos parcialmente observáveis. Os elementos básicos do projeto já são familiares:

- Os modelos de transição e sensoriamento são representados por uma **rede bayesiana dinâmica** (DBN), como descrito no Capítulo 15.
- A rede bayesiana dinâmica é estendida com nós de decisão e utilidade, como os que são usados nas **redes de decisão** do Capítulo 16. O modelo resultante é chamado de **rede de decisão dinâmica** ou DDN.
- Um algoritmo de filtragem é usado para incorporar cada nova percepção e cada nova ação, e para atualizar a representação de estados de crença.
- As decisões são tomadas projetando-se para frente sequências de ações diretas possíveis e escolhendo-se a melhor.

DBNs são **representações fatoradas** na terminologia do Capítulo 2, elas normalmente têm a vantagem da complexidade exponencial sobre as representações atômicas e podem modelar muitos problemas do mundo real. O projeto de agente é, portanto, uma implementação prática do **agente baseado na utilidade** esboçado no Capítulo 2.

Numa DBN, o único estado S_t torna-se um conjunto de variáveis de estado $\mathbf{X}_t$, e pode haver múltiplas variáveis de evidência $\mathbf{E}_t$. Usaremos A_t para fazer referência à ação no instante t e, assim, o modelo de transição se torna $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, A_t)$ e o modelo de sensoriamento se torna, $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$. Usaremos

R_t para fazer referência à recompensa recebida no instante t e U_t para fazer referência à utilidade do estado no tempo t (ambas são variáveis aleatórias). Com essa notação, uma rede de decisão dinâmica será semelhante à rede mostrada na Figura 17.10.

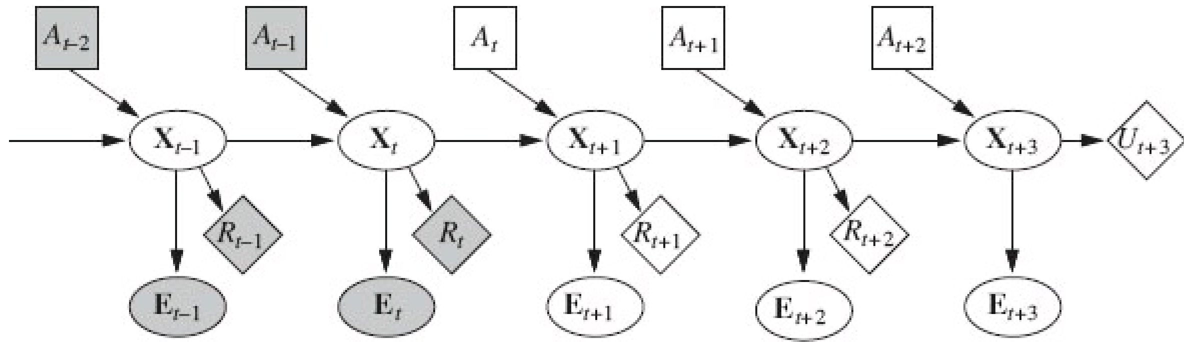


Figura 17.10 Estrutura genérica de uma rede de decisão dinâmica. As variáveis com valores conhecidos estão sombreadas. O tempo atual é t e o agente deve decidir o que fazer, isto é, escolher um valor para A_t. A rede foi desenvolvida para o futuro correspondente a três passos e representa recompensas futuras, bem como a utilidade do estado no horizonte de observação para frente.

As redes de decisão dinâmica podem ser usadas como entrada para qualquer algoritmo de POMDP, incluindo aqueles para métodos de iteração de valor e de política. Nesta seção, vamos nos concentrar em métodos de observação antecipada que projetam para frente sequências de ações, a partir do estado de crença atual, de maneira muito semelhante à forma como funcionam os algoritmos de jogos do Capítulo 5. A rede da Figura 17.10 foi projetada três passos para o futuro; as decisões atuais e futuras A, as observações futuras $\mathbf{E}$ e as recompensas futuras R são desconhecidas. Note que a rede inclui nós para as *recompensas* correspondentes a $\mathbf{X}_{t+1}$ e $\mathbf{X}_{t+2}$, mas a *utilidade* correspondente a $\mathbf{X}_{t+3}$. Isso ocorre porque o agente deve maximizar a soma (descontada) de todas as recompensas futuras, e $U(\mathbf{X}_{t+3})$ representa a recompensa para $\mathbf{X}_{t+3}$ e todas as recompensas subsequentes. Como no Capítulo 5, supomos que U só está disponível em alguma forma aproximada: se estivessem disponíveis valores de utilidade exata, não haveria necessidade de observação antecipada além da profundidade 1.

A Figura 17.11 mostra parte da árvore de busca que corresponde à DDN de observação antecipada de três passos da Figura 17.10. Cada um dos nós triangulares é um estado de crença em que o agente toma uma decisão A_{t+i} para $i = 0, 1, 2, \ldots$. Os nós (de chance) redondos correspondem a escolhas feitas pelo ambiente, isto é, em qual observação $\mathbf{E}_{t+i}$ ocorre. Note que não existe nenhum nó de chance correspondente aos resultados de ações; isso acontece porque a atualização do estado de crença para uma ação é determinística, não importando o resultado real.

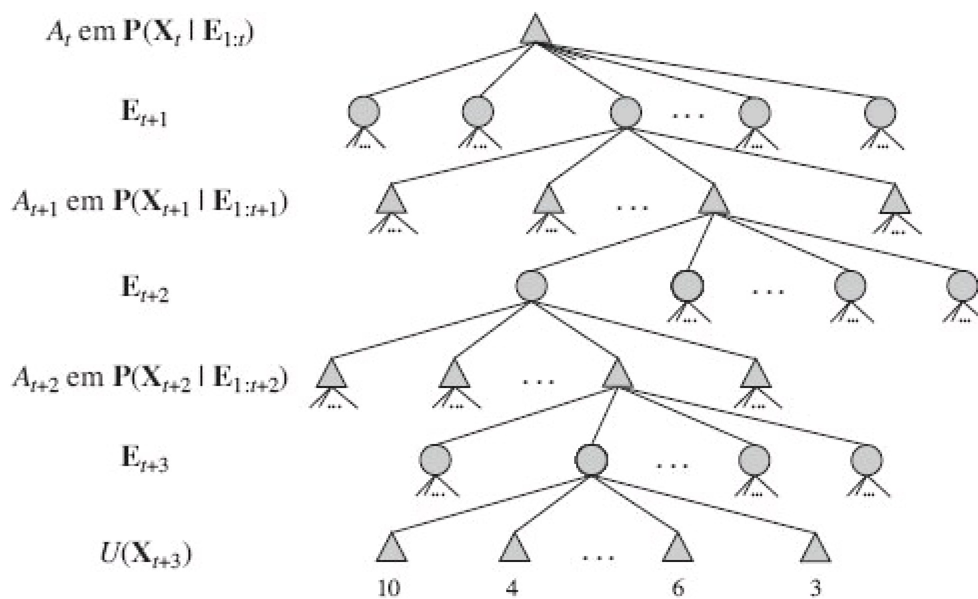


Figura 17.11 Parte da solução de observação antecipada da DDN da Figura 17.10. Cada decisão será tomada no estado de crença indicado.

O estado de crença em cada nó triangular pode ser calculado aplicando-se um algoritmo de filtragem à sequência de percepções e ações que levam a ele. Desse modo, o algoritmo leva em conta o fato de que, para a decisão A_{t+i}, o agente *terá* percepções disponíveis $\mathbf{E}_{t+1}, \ldots, \mathbf{E}_{t+i}$, embora, no tempo t, ele não saiba quais serão essas percepções. Desse modo, um agente de teoria da decisão levará em conta automaticamente o valor da informação e executará ações de coleta de informações onde for apropriado.

Uma decisão pode ser extraída da árvore de busca copiando-se os valores de utilidade das folhas, tomando uma média nos nós de chance e o máximo nos nós de decisão. Isso é semelhante ao algoritmo EXPECTIMINIMAX para árvores de jogos com nós de chance, exceto pelo fato de que (1) também podem existir recompensas em estados que não são folhas e (2) os nós de decisão correspondem a estados de crença, e não a estados reais. A complexidade de tempo de uma busca exaustiva até a profundidade d é $O(|A|^d \cdot |\mathbf{E}|^d)$, onde $|A|$ é o número de ações disponíveis e $|\mathbf{E}|$ é o número de percepções possíveis (note que é bem menos que o número de planos condicionais de profundidade d gerados pela iteração de valor). Para problemas em que o fator de desconto γ não é tão próximo de 1, uma busca rasa frequentemente é boa o bastante para gerar decisões quase ótimas. Também é possível fazer uma aproximação da etapa de cálculo da média nos nós de chance pela amostragem do conjunto de percepções possíveis, em vez de somar todas as percepções possíveis. Existem várias outras maneiras de descobrir rapidamente boas soluções aproximadas, mas vamos adiá-las até o Capítulo 21.

Os agentes de teoria da decisão baseados em redes de decisão dinâmicas têm diversas vantagens em comparação com outros projetos de agentes mais simples apresentados em capítulos anteriores. Em particular, eles tratam ambientes com incerteza, parcialmente observáveis, e podem revisar com facilidade seus "planos" para tratar percepções inesperadas. Com modelos apropriados de sensores, eles podem tratar falhas de sensores e planejar para obter informações. Esses agentes exibem "degradação suave" sob limitação de tempo e em ambientes complexos, utilizando várias técnicas de aproximação. Então, o que está faltando? Um defeito de nosso algoritmo baseado em DDN é sua confiança na busca para a frente através do espaço de estados, em vez de utilizar as técnicas de

planejamento hierárquicos e outras técnicas avançadas de planejamento descritas no Capítulo 11. Houve tentativas para estender essas técnicas ao domínio probabilístico, mas até agora elas se mostraram ineficientes. Um segundo problema relacionado é a natureza basicamente proposicional da linguagem de DDNs. Gostaríamos de poder estender algumas das ideias das linguagens probabilísticas de primeira ordem ao problema de tomada de decisões. A pesquisa atual tem mostrado que essa extensão é possível e tem benefícios significativos, como veremos nas notas no final do capítulo.

17.5 DECISÕES COM VÁRIOS AGENTES: TEORIA DOS JOGOS

Este capítulo se concentrou na tomada de decisões em ambientes incertos. Porém, e se a incerteza se dever a outros agentes e às decisões que eles tomam? E se as decisões desses agentes, por sua vez, forem influenciadas por nossas decisões? Já tratamos essa questão antes, quando estudamos os jogos no Capítulo 5. No entanto, naquele capítulo, estávamos preocupados com jogos de revezamento em ambientes completamente observáveis, para os quais a busca de minimax pode ser empregada para encontrar movimentos ótimos. Nesta seção, estudaremos os aspectos da **teoria dos jogos** que podem ser utilizados para analisar jogos com movimentos simultâneos e outras fontes de observabilidade parcial (os teóricos dos jogos usam as expressões **informação perfeita** e **informação imperfeita**, ao em vez de total e parcialmente observável). Existem pelo menos duas maneiras de utilizar a teoria dos jogos:

1. **Projeto de agentes**: A teoria dos jogos pode analisar as decisões do agente e calcular a utilidade esperada para cada decisão (sob a hipótese de que outros agentes estão agindo de forma ótima de acordo com a teoria dos jogos). Por exemplo, no jogo **par ou ímpar**, dois jogadores, *O* e *E*, exibem simultaneamente um ou dois dedos. Seja *f* o número total de dedos. Se *f* é ímpar, *O* recebe *f* dólares de *E* e, se *f* é par, *E* recebe *f* dólares de *O*. A teoria dos jogos pode determinar a melhor estratégia contra um jogador racional e o retorno esperado para cada jogador.[4]
2. **Projeto de mecanismos**: Quando um ambiente é habitado por muitos agentes, talvez seja possível definir as regras do ambiente (isto é, o jogo que os agentes devem jogar) de forma que o bem coletivo de todos os agentes seja maximizado quando cada agente adotar a solução da teoria dos jogos que maximiza sua própria utilidade. Por exemplo, a teoria dos jogos pode ajudar a projetar os protocolos para uma coleção de roteadores de tráfego da Internet, de forma que cada roteador tenha um incentivo para agir de tal modo que o *throughput* global seja maximizado. O projeto de mecanismos também pode ser usado para construir **sistemas multiagente** inteligentes que resolvem problemas complexos de modo distribuído.

17.5.1 Jogos de um movimento

Começamos por considerar um conjunto restrito de jogos: aqueles em que todos os jogadores agem simultaneamente e o resultado do jogo é baseado nesse único conjunto de ações. (Na verdade, não é

fundamental que as ações ocorram exatamente ao mesmo tempo; o que importa é que nenhum jogador tenha conhecimento das escolhas dos outros jogadores.) A restrição a um único movimento (e o próprio uso da palavra "jogo") pode fazer isso parecer trivial, mas na verdade a teoria dos jogos é um assunto sério.

Ela é utilizada em situações de tomada de decisão, incluindo o leilão de direitos de perfuração de petróleo e direitos de espectro de frequência sem fio, processos de falência, desenvolvimento de produtos e decisões de preços, e defesa nacional — situações que envolvem bilhões de dólares e centenas de milhares de vidas. Um jogo de um movimento é definido por três componentes:

- Os **jogadores** ou agentes que estarão tomando decisões. Os jogos de dois jogadores têm recebido mais atenção, embora também sejam comuns jogos de n jogadores, sendo $n > 2$. Utilizaremos nomes de jogadores com letras (iniciais) maiúsculas, como *Alice* e *Bob* ou *O* e *E*.
- As **ações** que os jogadores podem escolher. Empregaremos nomes em minúsculas para representar as ações, como *um* ou *testemunhar*. Os jogadores podem ter ou não o mesmo conjunto de ações disponíveis.
- Uma **função recompensa** que fornece a cada jogador a utilidade correspondente a cada combinação de ações realizadas por todos os jogadores. Para jogos de um movimento, a função recompensa pode ser representada por uma matriz, uma representação conhecida como **forma estratégica** (também chamada de **forma normal**). A matriz de recompensa para o jogo de par ou ímpar é dada a seguir:

	O: um	*O: dois*
E: um	$E = +2, O = -2$	$E = -3, O = +3$
E: dois	$E = -3, O = +3$	$E = +4, O = -4$

Por exemplo, o canto inferior direito mostra que, quando *O* escolhe a ação *dois* e *E* também escolhe *dois*, a recompensa é +4 para *E* e –4 para *O*.

Cada jogador em um jogo deve adotar e depois executar uma **estratégia** (o nome usado em teoria dos jogos para indicar uma *política*). Uma **estratégia pura** é uma política determinística; no caso de um jogo de um movimento, uma estratégia pura é apenas uma ação única. Para muitos jogos, um agente pode fazer melhor com uma **estratégia mista**, que é uma política aleatória que seleciona ações de acordo com uma distribuição de probabilidade. A estratégia mista que escolhe a ação a com probabilidade p e a ação b em caso contrário é escrita como $[p: a; (1 - p): b]$. Por exemplo, uma estratégia mista para par ou ímpar poderia ser [0,5: *um*; 0,5: *dois*]. Um **perfil de estratégia** é uma atribuição de uma estratégia para cada jogador; dado o perfil de estratégia, o **resultado** do jogo é um valor numérico para cada jogador.

Uma **solução** para um jogo é um perfil de estratégia em que cada jogador adota uma estratégia racional. Veremos que a questão mais importante em teoria dos jogos é definir o que significa "racional" quando cada agente escolhe apenas uma parte do perfil de estratégia que determina o resultado. É importante perceber que esses resultados são resultados reais da participação em um jogo, enquanto soluções são construções teóricas utilizadas para analisar um jogo. Veremos que

alguns jogos só têm uma solução em estratégias mistas. Porém, isso não significa que um jogador deva literalmente adotar uma estratégia mista para ser racional.

Considere a situação a seguir: dois suspeitos de roubo, Alice e Bob, são pegos em flagrante próximo à cena de um roubo e são interrogados separadamente pela polícia. Um promotor oferece um acordo a cada um: se você testemunhar contra seu parceiro como o líder do roubo do anel, ficará livre por ter cooperado, enquanto seu parceiro pegará 10 anos de prisão. No entanto, se um testemunhar contra o outro, ambos terão cinco anos de pena. Alice e Bob também sabem que, se os dois se recusarem a depor, a pena será de apenas um ano para cada um, pelo delito mais leve de estar de posse de propriedade roubada. Agora, Alice e Bob estão diante do chamado **dilema do prisioneiro:** eles devem testemunhar ou recusar o acordo? Sendo agentes racionais, Alice e Bob querem, cada um, maximizar sua própria utilidade esperada. Vamos supor que Alice esteja insensivelmente desinteressada em relação ao destino do seu companheiro e, portanto, sua utilidade diminui proporcionalmente ao número de anos que ela passará na prisão, não importando o que acontecerá a Bob. Por sua vez, Bob se sente exatamente do mesmo modo. Para ajudá-los a alcançar uma decisão racional, os dois constroem a seguinte matriz de recompensa:

	Alice: testemunhar	*Alice: recusar*
Bob: testemunhar	$A = -5, B = -5$	$A = -10, B = 0$
Bob: recusar	$A = 0, B = -10$	$A = -1, B = -1$

Alice analisa a matriz de recompensa da seguinte maneira: "Suponha que Bob testemunhe. Nesse caso, eu pego cinco anos se testemunhar e 10 anos se não o fizer; portanto, nesse caso, testemunhar é melhor. Por outro lado, se Bob se recusar, eu consigo 0 se testemunhar e um ano se me recusar; logo, também nesse caso, testemunhar é melhor. Desse modo, em um ou outro caso, para mim é melhor testemunhar, de forma que isso é o que devo fazer."

Alice descobriu que *testemunhar* é uma **estratégia dominante** para o jogo. Dizemos que uma estratégia s para o jogador p **domina fortemente** a estratégia s' se o resultado correspondente a s é melhor para p que o resultado correspondente a s', considerando-se todas as escolhas de estratégias pelos outros jogadores. A estratégia s **domina fracamente** s' se s é melhor que s' em pelo menos um perfil de estratégia e não é pior em qualquer outro. Uma estratégia dominante é uma estratégia que domina todas as outras. É irracional executar uma estratégia fortemente dominada e irracional não executar uma estratégia dominante, se ela existir. Sendo racional, Alice escolhe a estratégia dominante. Precisamos apenas de um pouco mais de terminologia antes de continuar: dizemos que um resultado é **ótimo de Pareto**[5] se não há nenhum outro resultado preferido por todos os jogadores. Um resultado é **dominado de Pareto** por outro resultado se todos os jogadores preferem o outro resultado.

Se for inteligente e também racional, Alice continuará a raciocinar assim: a estratégia dominante de Bob também é testemunhar. Então, ele testemunhará e nós dois pegaremos cinco anos. Quando cada jogador tem uma estratégia dominante, a combinação dessas estratégias é chamada **equilíbrio de estratégia dominante**. Em geral, um perfil de estratégia forma um **equilíbrio** se nenhum jogador pode se beneficiar da troca de estratégias, dado que todos os outros jogadores se mantêm com a mesma estratégia. Um equilíbrio é essencialmente um **ótimo local** no espaço de políticas; ele é a parte

superior de um pico que desce ao longo de toda dimensão, onde uma dimensão corresponde às escolhas de estratégias de um jogador.

O matemático John Nash (1928−) provou que *todo jogo tem pelo menos um equilíbrio.* O conceito geral de equilíbrio é agora chamado de **equilíbrio de Nash** em sua honra. Certamente, uma estratégia dominante de equilíbrio é o equilíbrio de Nash (Exercício 17.16), mas alguns jogos têm equilíbrio de Nash mas nenhuma estratégia dominante.

O *dilema* no dilema do prisioneiro é que o resultado do ponto de equilíbrio é pior para ambos os jogadores que o resultado que eles obteriam se ambos se recusassem a testemunhar. Em outras palavras, o resultado para a solução de equilíbrio é dominada de Pareto pelo resultado (–1, –1) de (*recusar*, *recusar*). Existe algum modo de Alice e Bob chegarem ao resultado (–1, –1)? Certamente uma opção *permissível* para ambos é se recusar a testemunhar, mas é difícil ver como os agentes racionais podem chegar lá, dada a definição do jogo. Qualquer dos jogadores que preferir jogar *recusar* perceberá que seria melhor jogar *testemunhar*. Esse é o poder de atração de um ponto de equilíbrio. Teóricos de jogos concordam que estar em equilíbrio de Nash é uma condição necessária para uma solução, embora discordem se essa é uma condição suficiente.

É suficientemente fácil chegar à solução (*recusar, recusar*) se modificarmos o jogo. Por exemplo, poderíamos mudar para um **jogo repetido** em que os jogadores sabem que se encontrarão novamente. Ou os agentes podem ter crenças morais que encorajam cooperação e justiça. O que significa que eles têm uma função utilidade diferente, necessitando de uma matriz de recompensa diferente, tornando esse um jogo diferente. Veremos mais adiante que alterar os agentes para ter poderes computacionais limitados, em vez da habilidade de raciocinar de maneira absolutamente racional, pode resultar em resultados sem equilíbrio, pois pode informar a um agente que o outro tem racionalida limitada. Nesse caso, estamos considerando um jogo diferente do descrito anteriormente na matriz de recompensa.

Agora, vamos examinar um jogo que não tem nenhuma estratégia dominante. A Acme, uma empresa fabricante de hardware para videogames, tem de decidir se sua próxima máquina de jogos usará DVDs ou discos Blu-ray. Enquanto isso, o produtor de software de videogames Best precisa decidir se deve produzir seu próximo jogo em DVD ou Blu-ray. Os lucros para ambos serão positivos se eles concordarem e negativos se discordarem, como mostra a matriz de recompensa a seguir:

	Acme: bluray	*Acme: dvd*
Best: bluray	$A = +9$, $B = +9$	$A = –4$, $B = –1$
Best: dvd	$A = –3$, $B = –1$	$A = +5$, $B = +5$

Não existe nenhum equilíbrio de estratégia dominante para esse jogo, mas existem *dois* equilíbrios de Nash*: (bluray, bluray)* e (*dvd*, *dvd*). Sabemos que esses são equilíbrios de Nash porque, se um ou outro jogador mudar unilateralmente para uma estratégia diferente, esse jogador ficará em pior situação. Agora os agentes têm um problema: *existem várias soluções aceitáveis, mas*, *se cada agente escolher uma solução diferente, ambos os agentes sofrerão*. Como eles podem concordar sobre uma solução? Uma resposta é que ambos devem escolher a solução ótima de Pareto *(bluray, bluray)*; isto é, podemos restringir a definição de "solução" ao único equilíbrio de Nash

ótimo de Pareto, *desde que exista um*. Todo jogo tem pelo menos uma solução ótima de Pareto, mas um jogo poderia ter várias ou elas não seriam pontos de equilíbrio. Por exemplo, se (*bluray, bluray*) tiver recompensa (5,5), haverá dois pontos de equilíbrio ótimo de Pareto iguais entre si. Para escolher entre eles, os agentes podem adivinhar ou *se comunicar*, o que pode ser feito estabelecendo-se uma convenção que ordena as soluções antes de o jogo começar ou negociando para alcançar uma solução mutuamente benéfica durante o jogo (o que significaria incluir ações de comunicação como parte de um jogo sequencial). Desse modo, a comunicação surge na teoria dos jogos exatamente pelas mesmas razões que a fizeram surgir no planejamento de multiagentes na Seção 11.4. Jogos como esse em que os jogadores precisam se comunicar são chamados de **jogos de coordenação**.

Vimos que um jogo pode ter mais de um equilíbrio de Nash; como sabemos que todo jogo deve ter pelo menos um? Alguns jogos não têm nenhum equilíbrio de Nash de *estratégia pura*. Por exemplo, considere qualquer perfil de estratégia pura para o jogo de par ou ímpar. Se o número total de dedos for par, O desejará trocar sua aposta; por outro lado, se o total for ímpar, então E desejará mudar. Portanto, nenhum perfil de estratégia pura pode ser um equilíbrio e nesse caso devemos examinar estratégias mistas. Mas qual estratégia mista? Em 1928, von Neumann desenvolveu um método para encontrar a *estratégia mista ótima* para **jogos de soma zero** — jogos em que a soma das recompensas é sempre zero.[6] Sem dúvida, o jogo de par ou ímpar é um desses jogos. No caso de jogos de dois jogadores de soma zero, sabemos que as recompensas são iguais e opostas, e portanto precisamos considerar as recompensas de apenas um jogador, que será o maximizador (da mesma maneira que no Capítulo 5). No caso do par ou ímpar, selecionamos o jogador que escolhe par, E, para ser o maximizador, e então podemos definir a matriz de recompensa pelos valores $U_E(e, o)$ — a recompensa para E se E executa a ação e e O executa o (por conveniência, chamaremos o jogador E de "sua" e o jogador O de "seu"). O método de von Neumann é chamado de técnica de **maximin**, e funciona da maneira descrita a seguir:

- Suponha que mudamos as regras da seguinte forma: primeiro E escolhe sua estratégia e revela para O. Então, O escolhe sua estratégia, com conhecimento da estratégia de E. Finalmente, avaliamos a recompensa esperada do jogo com base nas estratégias escolhidas. Isso nos dá um jogo de revezamento ao qual podemos aplicar o algoritmo **minimax** do Capítulo 5. Vamos supor que isso forneça um resultado $U_{E,O}$. É claro que esse jogo favorece O, e, assim, a verdadeira utilidade U do jogo original (do ponto de vista de E) é, *no mínimo*, $U_{E,O}$. Por exemplo, se simplesmente examinarmos estratégias puras, a árvore de jogo de minimax terá um valor de raiz igual a –3 (veja a Figura 17.12(a)), e assim sabemos que $U \geq -3$.

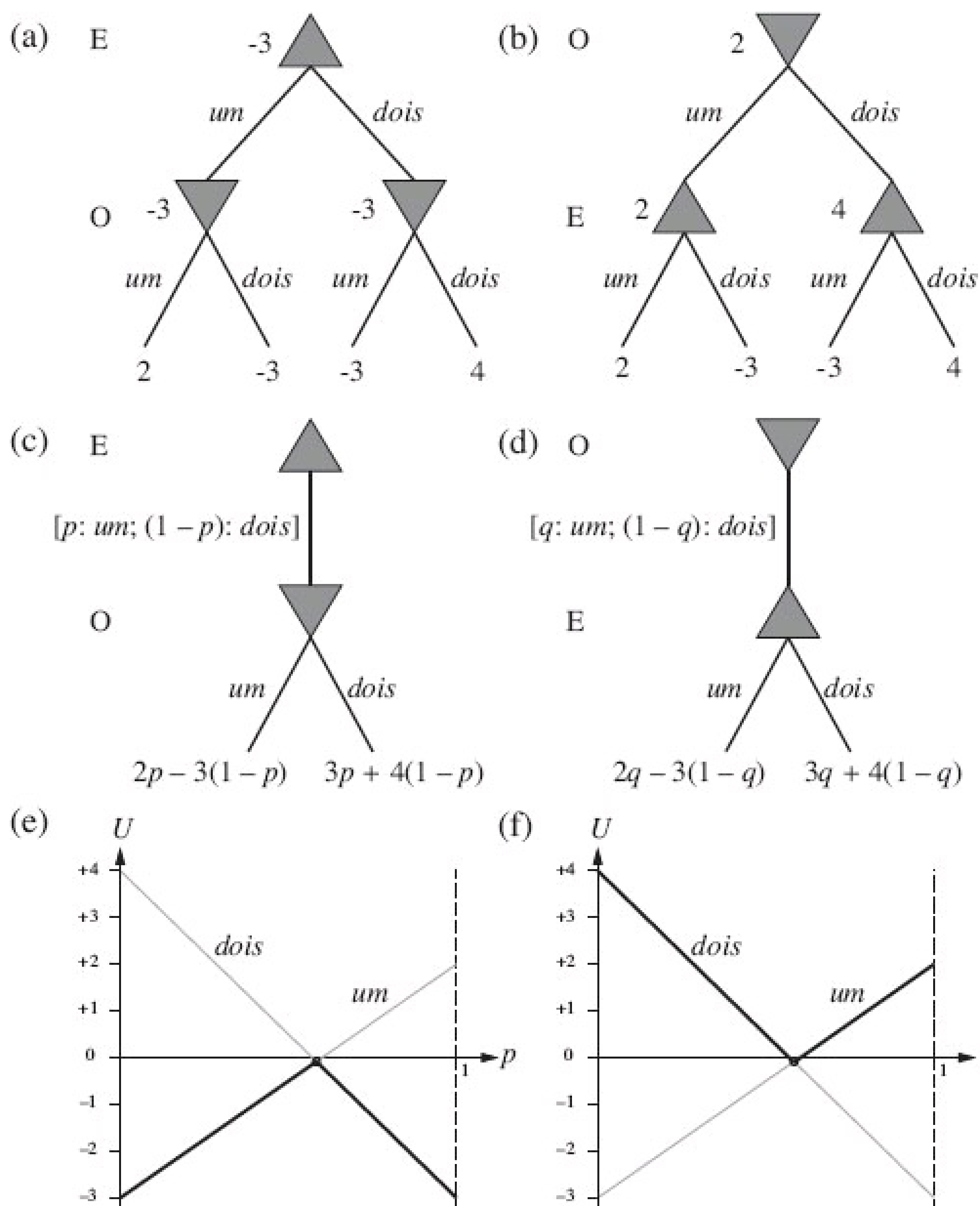


Figura 17.12 (a) e (b): Árvores de jogos de minimax para o jogo par ou ímpar, se os jogadores se revezam jogando estratégias puras. (c) e (d): Árvores de jogos parametrizadas em que o primeiro jogador executa uma estratégia mista. As recompensas dependem do parâmetro de probabilidade (*p* ou *θ*) na estratégia mista. (e) e (f): Para qualquer valor específico do parâmetro de probabilidade, o segundo jogador escolherá a "melhor" das duas ações e, assim, o valor da estratégia mista do primeiro jogador é dado pelas linhas grossas. O primeiro jogador escolherá o parâmetro de probabilidade para a estratégia mista no ponto de interseção.

- Agora, vamos supor que alteramos as regras para forçar *O* a revelar sua estratégia primeiro, seguido por *E*. Então, o valor de minimax desse jogo será $U_{O,E}$ e, como esse jogo favorece *E*, sabemos que *U* é, *no máximo*, $U_{O,E}$. Com estratégias puras, o valor é +2 (veja a Figura 17.12(b)), e assim sabemos que $U \leq +2$.

Combinando esses dois argumentos, vemos que a verdadeira utilidade *U* da solução para o jogo original deve satisfazer

$$U_{E,O} \leq U \leq U_{O,E} \quad \text{ou, nesse caso,} \quad -3 \leq U \leq 2\,.$$

Para definir o valor de U, precisamos deslocar nossa análise para as estratégias mistas. Primeiro, observe: *uma vez que o primeiro jogador revela sua estratégia, o segundo jogador pode também escolher uma estratégia pura*. A razão é simples: se o segundo jogador jogar uma estratégia mista, [p: *um*; $(1 - p)$: *dois*], sua utilidade esperada será uma combinação linear $(p \cdot u_{um} + (1 - p) \cdot u_{dois})$ das utilidades das estratégias puras u_{um} e u_{dois}. Essa combinação linear nunca pode ser melhor que a melhor entre u_{um} e u_{dois}, e assim o segundo jogador pode simplesmente escolher a melhor delas.

Com essa observação em mente, as árvores de minimax podem ser imaginadas como tendo número infinito de ramificações na raiz, correspondentes ao número infinitamente grande de estratégias mistas que o primeiro jogador pode escolher. Cada uma delas leva a um nó com duas ramificações que correspondem às estratégias puras para o segundo jogador. Podemos representar essas árvores infinitas de forma finita, tendo uma única escolha "parametrizada" na raiz:

- Se E se move primeiro, a situação é a mostrada na Figura 17.12(c). E escolhe a estratégia [p: *um*; $(1–p)$: *dois*] na raiz, e depois O escolhe uma estratégia pura (e por isso um movimento) dado o valor de p. Se O escolher *um*, a recompensa esperada (para E) é $2p - 3(1 - p) = 5p - 3$; se O escolher *dois*, a recompensa esperada é $-3p + 4(1 - p)= 4 - 7p$. Podemos desenhar essas duas recompensas como linhas retas em um grafo, onde p varia de 0 até 1 no eixo x, como mostra a Figura 17.12(e). O, o minimizador, sempre escolherá a mais baixa das duas linhas, como mostram as linhas grossas na figura. Portanto, o melhor que E pode fazer na raiz é escolher p para o ponto de interseção, onde:

$$5p - 3 = 4 - 7p \Rightarrow p = 7/12.$$

A utilidade para E nesse ponto é $U_{E,O} = -1/12$.

- Se O se mover primeiro, a situação será a da Figura 17.12(d). O escolhe a estratégia [θ: *um*; $(1-\theta)$: *dois*] na raiz, e então E escolhe um movimento, dado o valor de θ. As recompensas são $2\theta - 3(1 - \theta) = 5\theta - 3$ e $-3\theta + 4(1 - \theta) = 4 - 7\theta$.[7] Novamente, a Figura 17.12(f) mostra que o melhor que O pode fazer na raiz é escolher o ponto de interseção:

$$5\theta - 3 = 1 - 7\theta \Rightarrow \theta = 7/12.$$

A utilidade para E nesse ponto é $U_{O,E} = -1/12$.

Agora sabemos que a verdadeira utilidade do jogo original fica entre –1/12 e –1/12, ou seja, ela é exatamente –1/12! (A moral é que é melhor ser O do que E, se você estiver participando desse jogo.) Além disso, a verdadeira utilidade é atingida pela estratégia mista [7/12: *um*; 5/12: *dois*], que deve ser utilizada por ambos os jogadores. Essa estratégia é chamada de **equilíbrio de maximin** do jogo, e é um equilíbrio de Nash. Observe que cada estratégia componente em uma estratégia de equilíbrio misto tem a mesma utilidade esperada. Nesse caso, tanto *um* quanto *dois* têm a mesma utilidade esperada, –1/12, como a estratégia mista propriamente dita.

Nosso resultado para o jogo par ou ímpar é um exemplo do resultado geral enunciado por von

Neumann: *todo jogo de soma zero de dois jogadores tem equilíbrio de maximin quando se permitem estratégias mistas*. Além disso, todo equilíbrio de Nash em um jogo de soma zero é um maximin para ambos os jogadores. Um jogador que adota a estratégia de maximin tem duas garantias: em primeiro lugar, nenhuma outra estratégia pode ser melhor contra um adversário que jogue bem (apesar de que algumas outras estratégias podem ser melhores em explorar um adversário que comete erros irracionais). Em segundo lugar, o jogador continua a jogar bem, mesmo que a estratégia seja revelada ao adversário.

O algoritmo geral para encontrar equilíbrios de maximin em jogos de soma zero é um pouco mais complicado do que as Figuras 17.12(e) e (f) poderiam sugerir. Quando existem *n* ações possíveis, uma estratégia mista é um ponto em um espaço *n* dimensional, e as linhas se tornam hiperplanos. Também é possível que algumas estratégias puras para o segundo jogador sejam dominadas por outras, de modo que elas não são ótimas contra *qualquer* estratégia para o primeiro jogador. Depois de remover todas essas estratégias (o que talvez tenha de ser feito repetidamente), a escolha ótima na raiz é o mais alto (ou mais baixo) ponto de interseção dos hiperplanos restantes. A descoberta dessa escolha é um exemplo de problema de **programação linear**: maximizar uma função objetivo, sujeita a restrições lineares. Tais problemas podem ser resolvidos por técnicas padrões em tempo polinomial no número de ações (e no número de bits usados para especificar a função recompensa, se quisermos ser mais técnicos).

A pergunta permanece: o que um agente racional realmente deve *fazer* para jogar uma única partida de par ou ímpar? O agente racional terá derivado o fato de que [7/12: *um*; 5/12: *dois*] é a estratégia de equilíbrio de maximin e assumiremos que esse seja um conhecimento mútuo com um oponente racional. O agente poderia usar um dado de 12 faces ou um gerador de números aleatórios para escolher ao acaso de acordo com essa estratégia mista e, nesse caso, a recompensa esperada seria –1/12 para *E*. Ou, então, o agente poderia simplesmente decidir jogar *um* ou *dois*. Em um ou outro caso, a recompensa esperada permanece –1/12 para *E*. Curiosamente, a escolha unilateral de uma ação específica não prejudica a recompensa esperada de nenhum dos dois agentes, mas permitir ao outro agente saber que se tomou tal decisão unilateral *afeta* a recompensa esperada porque o oponente poderá ajustar sua estratégia de acordo com isso.

Descobrir o equilíbrio em jogos de soma diferentes de zero é um pouco mais complicado. A abordagem geral tem duas etapas: (1) enumerar todos os possíveis subconjuntos de ações que poderiam formar estratégias mistas. Por exemplo, primeiro tente todos os perfis de estratégias em que cada jogador usa uma única ação, depois aqueles em que cada jogador utiliza uma ou duas ações, e assim por diante. Isso é exponencial em relação ao número de ações e, assim, só se aplica a jogos relativamente pequenos. (2) Para cada perfil de estratégia enumerada em (1), verificar se ele é um equilíbrio. Isso é feito resolvendo-se um conjunto de equações e desigualdades semelhantes às que foram usadas no caso de soma zero. Para dois jogadores, essas equações são lineares e podem ser resolvidas com técnicas básicas de programação linear; porém, para três ou mais jogadores, elas são não lineares, e pode ser muito difícil resolvê-las.

17.5.2 Jogos repetidos

Até agora, examinamos apenas jogos que duram um único movimento. A espécie mais simples de jogo de vários movimentos é o **jogo repetido**, no qual os jogadores ficam diante da mesma escolha repetidamente, mas, em cada vez, com conhecimento da história das escolhas anteriores de todos os jogadores. Um perfil de estratégia para um jogo repetido especifica uma escolha de ação para cada jogador em cada período de tempo para toda história possível de escolhas anteriores. Como ocorre com MDPs, as recompensas são aditivas com o passar do tempo.

Vamos considerar a versão repetida do dilema do prisioneiro. Alice e Bob se recusarão a testemunhar, sabendo que se encontrarão novamente? A resposta depende dos detalhes do compromisso. Por exemplo, suponha que Alice e Bob saibam que devem jogar exatamente 100 rodadas do dilema do prisioneiro. Então, ambos sabem que a 100ª rodada não será um jogo repetido — isto é, seu resultado pode não ter nenhum efeito sobre rodadas futuras — e, portanto, ambos escolherão a estratégia dominante, *testemunhar*, nessa rodada. Porém, uma vez que a 100ª rodada seja determinada, a 99ª rodada poderá não ter nenhum efeito sobre rodadas subsequentes e, assim, ela também terá um equilíbrio de estratégia dominante em (*testemunhar*, *testemunhar*). Por indução, ambos os jogadores escolherão *testemunhar* em toda rodada, ganhando cada um uma sentença de prisão total de 500 anos.

Podemos obter diferentes soluções alterando as regras da interação. Por exemplo, suponha que, depois de cada rodada, exista uma chance de 99% dos jogadores se encontrarem novamente. Então, o número esperado de rodadas ainda será 100, mas nenhum jogador saberá com certeza qual rodada será a última. Sob essas condições, é possível um comportamento mais cooperativo. Por exemplo, uma estratégia de equilíbrio é cada jogador *recusar*, a menos que o outro jogador tenha jogado *testemunhar*. Essa estratégia poderia ser chamada de **castigo perpétuo**. Suponha que ambos os jogadores tenham adotado essa estratégia e que ela seja de conhecimento mútuo. Então, desde que nenhum jogador tenha jogado *testemunhar*, em qualquer instante a recompensa total esperada no futuro para cada jogador será:

$$\sum_{t=0}^{\infty} 0{,}99^t \cdot (-1) = -100 \, .$$

Um jogador que desvia da estratégia e escolhe *testemunhar* ganhará uma pontuação 0 em lugar de –1 em seu próximo movimento, mas daí em diante os dois jogadores vão jogar *testemunhar* e a recompensa futura esperada total do jogador se tornará:

$$0 + \sum_{t=1}^{\infty} 0{,}99^t \cdot (-5) = -495 \, .$$

Então, em cada etapa, não haverá nenhum incentivo para divergir de (*recusar*, *recusar*). O castigo perpétuo é a estratégia de "destruição mutuamente garantida" do dilema do prisioneiro: uma vez que um ou outro jogador decide testemunhar, ela assegura que ambos os jogadores sofrerão muito. No entanto, ela só funcionará como um meio de intimidação se o outro jogador acreditar que você adotou essa estratégia — ou, pelo menos, que você poderia tê-la adotado.

Existem outras estratégias mais generosas. A mais famosa, chamada **tête-à-tête** (tit-for-tat), consiste em começar com *recusar* e depois ecoar o movimento anterior do outro jogador em todos os

movimentos subsequentes. Assim, Alice se recusaria desde que Bob se recusasse e testemunharia no movimento seguinte ao testemunho de Bob, mas voltaria a recusar se Bob o fizesse. Embora muito simples, essa estratégia se mostrou altamente robusta e eficiente contra uma ampla variedade de estratégias.

Também podemos obter diferentes soluções alterando os agentes, em vez de alterar as regras de compromisso. Suponha que os agentes sejam máquinas de estados finitos com n estados e que estejam participando de um jogo com $m > n$ passos no total. Desse modo, os agentes são incapazes de representar o número de passos restantes e devem tratá-lo como uma incógnita. Então, eles não podem realizar a indução e ficam livres para chegar ao equilíbrio mais favorável (*recusar*, *recusar*). Nesse caso, ignorância *é* felicidade, ou melhor, fazer seu oponente acreditar que você é ignorante é a felicidade. Seu sucesso nesses jogos repetidos depende da *percepção* do outro jogador de que você é um tirano ou um simplório, e não de suas características reais.

17.5.3 Jogos sequenciais

Em geral, um jogo consiste em uma sequência de turnos que não precisam ser todos iguais. Tais jogos são mais bem representados por uma árvore de jogo, que os teóricos dos jogos chamam de **forma extensiva**. A árvore inclui todas as mesmas informações que vimos na Seção 5.1: um estado inicial S_0, uma função JOGADOR(s), que diz que jogador tem a vez, uma função AÇÕES(s), que enumera as ações possíveis, uma função RESULTADO(s, a), que define a transição para um novo estado, e uma função parcial UTILIDADE(s, p), que é definida apenas em estados terminais, para dar a recompensa para cada jogador.

Para representar jogos estocásticos, como gamão, adicionamos um jogador distinto, *chance*, que pode tomar medidas aleatórias. A "estratégia" do acaso é parte da definição do jogo, especificado como uma distribuição de probabilidade sobre as ações (os outros jogadores escolhem sua própria estratégia). Para representar jogos com ações não determinísticas, como o bilhar, separamos a ação em duas partes: a ação do jogador em si tem resultado determinístico e, depois, *chance* tem a vez para reagir à ação em sua própria maneira caprichosa. Para representar movimentos simultâneos, como no dilema do prisioneiro ou no jogo de par ou ímpar, impomos uma ordem arbitrária aos jogadores, mas temos a opção de afirmar que as ações anteriores do jogador não são observáveis aos jogadores subsequentes: por exemplo, Alice deve escolher *recusar* ou *depor* primeiro, então Bob escolhe, mas Bob não sabe qual a escolha de Alice na sua vez (podemos também representar o fato de que o movimento será revelado mais tarde). No entanto, assumimos que os jogadores sempre lembram todos as suas *próprias* ações anteriores; essa suposição é chamada de **memória perfeita**.

A ideia-chave da forma extensiva que a distingue das árvores de jogo do Capítulo 5 é a representação de observabilidade parcial. Vimos na Seção 5.6 que um jogador em um jogo parcialmente observável, como o Kriegspiel, pode criar uma árvore de jogo sobre o espaço de **estados de crença**. Com essa árvore, vimos que, em alguns casos, um jogador pode encontrar uma sequência de movimentos (uma estratégia) que leva a um xeque-mate forçado, independentemente de qual estado real iniciamos, e independentemente de qual estratégia o adversário usou. No entanto, as técnicas do Capítulo 5 não poderiam informar a um jogador o que fazer quando não houver um xeque-

mate garantido. Se a melhor estratégia do jogador depende da estratégia do adversário e vice-versa, o minimax (ou alfa-beta), não pode encontrar uma solução por si só. A forma extensiva permite que encontremos soluções porque representa o estado de crença (teóricos dos jogos chamam de **conjuntos de informações**) de todos os jogadores de uma vez. Dessa representação, podemos encontrar soluções de equilíbrio, assim como fizemos com a forma normal de jogos.

Como exemplo simples de um jogo sequencial, coloque dois agentes no mundo 4 × 3 da Figura 17.1 e faça-os moverem-se simultaneamente até que um agente chegue a um quadrado de saída e pegue a recompensa desse quadrado. Se especificarmos que não ocorre nenhum movimento quando os dois agentes tentarem mover-se no mesmo quadrado simultaneamente (um problema comum nos cruzamentos de trânsito), determinadas estratégias puras podem ficar presas para sempre. Assim, os agentes precisam de uma estratégia mista para um bom desempenho nesse jogo: escolher aleatoriamente entre avançar e permanecer no lugar. Isso é exatamente o que é feito para resolver colisões de pacotes em redes Ethernet.

Em seguida, vamos considerar uma variante muito simples do pôquer. O baralho tem apenas quatro cartas, dois ases e dois reis. Uma carta é distribuída a cada jogador. O primeiro jogador tem a opção de *aumentar* a aposta do jogo a partir de 1 ponto para 2 ou *passar*. Se o jogador 1 passar, o jogo termina. Se ele aumentar a aposta, o jogador 2 tem a opção de *igualar o valor da aposta* aceitando que o jogo vale 2 pontos ou *desistir da mão*, concedendo 1 ponto. Se o jogo não terminar com a desistência, a recompensa depende das cartas: zero para ambos os jogadores se eles tiverem a mesma carta; caso contrário, o jogador com o rei paga a aposta para o jogador com o ás.

A árvore de forma extensiva para esse jogo é mostrada na Figura 17.13. Os estados não terminais são mostrados como círculos, com o jogador a se mover dentro do círculo; o jogador 0 é *chance*. Cada ação é retratada como uma seta com um rótulo, correspondente a *aumentar a aposta*, *passar*, *igualar o valor da aposta*, *desistir da mão ou arriscar* as quatro jogadas possíveis ("AK" significa que o jogador 1 recebeu um ás, e o jogador 2, um rei). Estados terminais são retângulos rotulados pelas suas recompensas para os jogadores 1 e 2. Os conjuntos de informações são mostrados como caixas tracejadas rotuladas, por exemplo, $I_{1,1}$ é o conjunto de informações, em que é a vez do jogador 1, e ele sabe que tem um ás (mas não sabe o que o jogador 2 tem). No conjunto de informação $I_{2,1}$, é a vez do jogador 2, e ele sabe que tem um ás e o jogador 1 aumentou a aposta, mas não sabe que cartão o jogador 1 tem (devido ao limite do papel bidimensional, esse conjunto de informação é mostrado como duas caixas em vez de uma).

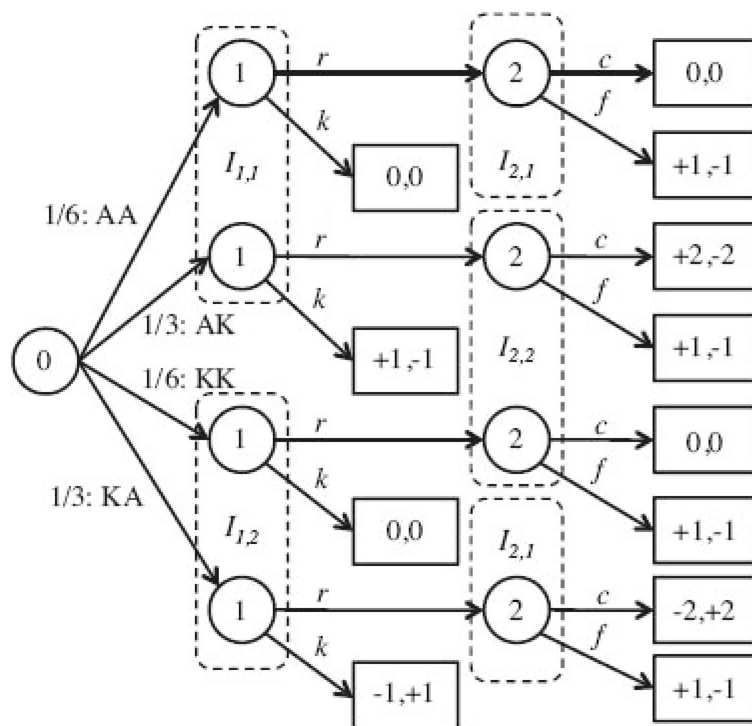


Figura 17.13 Forma extensiva de uma versão simplificada do pôquer.

Uma maneira de resolver um jogo extensivo é convertê-lo para um jogo de forma normal. Lembre-se de que a forma normal é uma matriz, cada linha é rotulada com uma estratégia pura para o jogador 1 e cada coluna por uma estratégia pura para o jogador 2. Em um jogo extensivo, uma estratégia pura para o jogador *i* corresponde a uma ação para cada conjunto de informações envolvendo esse jogador. Assim, na Figura 17.13, uma estratégia pura para o jogador 1 é "aumentar a aposta quando em $I_{1,1}$ (isto é, quando tiver um ás) e passar quando em $I_{1,2}$ (quando tiver um rei). Na matriz de recompensa a seguir, essa estratégia é chamada de *rk*. Da mesma forma, a estratégia *cf* do jogador 2 significa "igualar o valor da aposta quando eu tiver um ás e desistir da mão quando tiver um rei". Como esse é um jogo de soma zero, a matriz a seguir dá apenas a recompensa para o jogador 1; o jogador 2 tem sempre a recompensa oposta:

	2:*cc*	2:*cf*	2:*ff*	2:*fc*
1:rr	0	−1/6	1	7/6
1:*kr*	−1/3	−1/6	5/6	2/3
1:rk	1/3	**0**	1/6	1/2
1:*kk*	0	**0**	0	0

Esse jogo é tão simples que tem dois equilíbrios de estratégia pura, mostrador em negrito: *cf* para o jogador 2 e *rk* ou *kk* para o jogador 1. Mas, em geral, podemos resolver jogos extensivos, convertendo para a forma normal e depois encontrando uma solução (geralmente uma estratégia mista), utilizando métodos de programação linear padrão. Isso funciona na teoria. Mas, se um jogador tiver conjuntos de informação *I* e ações *a* por conjunto, esse jogador terá a^I estratégias puras. Em outras palavras, o tamanho da matriz de forma normal é exponencial em número de conjuntos de informações, então na prática a abordagem funciona apenas para árvores de jogo muito pequenas, da ordem de uma dúzia de estados. Um jogo como o Texas Hold'em Poker tem cerca de 10^{18} estados,

tornando essa abordagem completamente inviável.

Quais são as alternativas? No Capítulo 5, vimos como a busca alfa-beta poderia lidar com jogos de informação perfeita com árvores de jogo enormes, através da geração da árvore de jogo de forma incremental, com a poda de algumas ramificações e a avaliação heurística de nós não terminais. Mas essa abordagem não funciona bem para jogos com informação imperfeita, por duas razões: primeiro, é mais difícil podar porque precisamos considerar estratégias mistas que combinem várias ramificações, não uma estratégia pura que sempre escolha a melhor ramificação. Segundo, é mais difícil avaliar heuristicamente um nó não terminal porque estamos lidando com conjuntos de informação, não com estados individuais.

Koller *et al*. (1996) vieram em auxílio com uma representação alternativa de jogos extensivos, chamada de **forma sequencial**, que é apenas linear no tamanho da árvore, em vez de exponencial. Em vez de representar estratégias, representa caminhos através da árvore; o número de caminhos é igual ao número de nós terminais. Métodos de programação linear padrão podem ser aplicados novamente para essa representação. O sistema resultante pode resolver variantes do pôquer com 25.000 estados em um minuto ou dois. Essa é uma aceleração exponencial sobre a abordagem da forma normal, mas ainda está muito aquém do tratamento do pôquer completo, com 10^{18} estados.

Se não podemos lidar com 10^{18} estados, talvez possamos simplificar o problema, alterando o jogo para uma forma mais simples. Por exemplo, se eu segurar um ás considerando a possibilidade de que próxima carta vá me dar um par de ases, então não importa o naipe da próxima carta, qualquer naipe serve. Isso sugere a formação de uma **abstração** do jogo, em que os naipes são ignorados. A árvore de jogo resultante será menor por um fator de 4! = 24. Supondo que eu possa resolver esse pequeno jogo, como é que a solução para esse jogo se relaciona com o jogo original? Se nenhum jogador estiver seguindo para um *flush* (cinco cartas do mesmo naipe) (ou um blefe), então os naipes não importam para nenhum jogador, e a solução para a abstração será também uma solução para o jogo original. No entanto, se houver qualquer jogador que esteja contemplando um *flush*, a abstração será apenas uma solução aproximada (mas é possível calcular os limites sobre o erro).

Existem muitas oportunidades para a abstração. Por exemplo, em um jogo, no ponto em que cada jogador tem duas cartas, se eu tiver um par de damas, a mão dos outros jogadores pode ser abstraída em três classes: *melhor* (apenas um par de reis ou um par de ases), *mesma* (par de rainhas) ou *pior* (o restante). No entanto, essa abstração pode ser muito tosca. Uma melhor abstração seria dividir *pior* em, digamos, *par médio* (nove até o valete), *par baixo* e *nenhum par*. Esses exemplos são abstrações de estados, sendo possível também abstrair ações. Por exemplo, em vez de ter uma ação de aposta para cada inteiro de 1 a 1.000, poderíamos restringir as apostas para 10^0, 10^1, 10^2 e 10^3. Ou poderíamos cortar completamente uma das rodadas de apostas. Podemos também abstrair sobre nós de chance, considerando apenas um subconjunto de tratos possíveis. Isso é equivalente à técnica de *rollout* usada em programas Go. Colocando todas essas abstrações juntas, podemos reduzir os 10^{18} estados do pôquer para 10^7 estados, um tamanho que pode ser resolvido com as técnicas atuais.

Os programas de pôquer com base nessa abordagem podem facilmente derrotar novatos e alguns jogadores humanos experientes, mas ainda não estão no nível de jogadores mestres. Parte do problema é que a solução aproximada desses programas — a solução de equilíbrio — é ótima apenas contra um adversário que também joga a estratégia de equilíbrio. Contra os jogadores humanos falíveis é importante ser capaz de explorar o desvio de um oponente da estratégia de

equilíbrio. Como Gautam Rao (também conhecido como "o conde"), líder mundial de jogo de pôquer on-line, disse (Billings *et al.*, 2003): "Você tem um programa muito forte. Depois de adicionar uma modelagem do adversário, ele vai matar todos." No entanto, bons modelos de jogadores humanos ainda precisam ser criados.

Em certo sentido, a forma extensiva do jogo é uma das representações mais completas que temos visto até agora: pode lidar com ambientes parcialmente observáveis, multiagentes, estocásticos, sequenciais, dinâmicos — a maioria dos casos difíceis da lista de propriedades do ambiente. No entanto, há duas limitações da teoria dos jogos. Primeiro, ela não lida bem com estados contínuos e ações (embora tenha havido algumas extensões para o caso contínuo; por exemplo, a teoria da **competição de Cournot** utiliza a teoria dos jogos para resolver problemas em que duas empresas escolhem os preços de seus produtos a partir de um espaço contínuo). Segundo a teoria dos jogos, assume que o jogo é *conhecido.* Partes do jogo podem ser especificadas para alguns dos jogadores como não observáveis, mas deve-se saber que partes são não observáveis. Em casos em que os jogadores aprendem a estrutura desconhecida do jogo ao longo do tempo, o modelo começa a entrar em colapso. Examinemos cada fonte de incerteza e se cada uma delas pode ser representada na teoria dos jogos.

Ações: não há nenhuma maneira fácil de representar um jogo em que os jogadores têm de descobrir que ações estão disponíveis. Considere o jogo entre os criadores de vírus de computador e os especialistas em segurança. Parte do problema é antecipar qual será a próxima ação dos criadores de vírus.

Estratégias: a teoria dos jogos é muito boa em representar a ideia de que a estratégia dos outros jogadores é inicialmente desconhecida — desde que assumamos que todos os agentes são racionais. A teoria em si não diz o que fazer quando outros jogadores são menos que totalmente racionais. A noção do **equilíbrio de Bayes-Nash** aborda parcialmente esse ponto: é um equilíbrio com relação à distribuição da probabilidade anterior de um jogador sobre as estratégias dos outros jogadores — em outras palavras, expressa as crenças de um jogador sobre as estratégias prováveis dos outros jogadores.

Chance: Se um jogo depende do lançamento de um dado, será bem fácil modelar um nó de acaso com distribuição uniforme sobre os resultados. Mas, e se for possível que o dado seja viciado? Podemos representar isso com outro nó de acaso, mais acima na árvore, com duas ramificações para "dado imparcial" e "dado viciado", de tal forma que os nós correspondentes em cada ramificação estão no mesmo conjunto de informação (isto é, os jogadores não sabem se o dado é viciado ou não). E se suspeitarmos que o outro adversário sabe? Então adicionamos *outro* nó de acaso, com uma ramificação representando o caso em que o adversário sabe, e um em que ele não sabe.

Utilidades: e se não conhecermos as utilidades de nosso oponente? Mais uma vez, pode ser modelado com um nó de acaso, de tal forma que o outro agente conheça a própria utilidade em cada ramificação, mas nós, não. E se não conhecermos nossa própria utilidade? Por exemplo, como sei se é racional pedir a salada do chefe, se não sei o quanto vou gostar? Podemos modelar isso com mais um nó de acaso especificando uma "qualidade intrínseca" não observável da salada.

Assim, vemos que a teoria dos jogos é eficaz em representar a maioria das origens de incerteza, mas ao custo de dobrar o tamanho da árvore cada vez que adicionamos outro nó, um hábito que leva rapidamente a árvores grandes que se tornam intratáveis. Devido a esses e outros problemas da

teoria dos jogos, usa-se a princípio *analisar* ambientes que estão em equilíbrio, em vez de *controlar* agentes dentro de um ambiente. Em seguida veremos como o *projeto* de ambientes pode ajudar.

17.6 PROJETO DE MECANISMOS

Na seção anterior, perguntamos: "Dado um jogo, o que é uma estratégia racional?" Nesta seção, perguntamos: "Dados os agentes que escolhem estratégias racionais, que jogo devemos projetar?" Mais especificamente, gostaríamos de projetar um jogo cujas soluções, consistindo na busca por cada agente de sua própria estratégia racional, resultassem na maximização de alguma função utilidade global. Esse problema é chamado de **projeto de mecanismos** ou, às vezes, **teoria dos jogos inversa**. O projeto de mecanismos é uma mistura de economia e ciência política. O capitalismo 101 diz que, se todos tentam ficar ricos, a riqueza total da sociedade vai aumentar. Mas os exemplos que vamos discutir mostram que é necessário um projeto de mecanismo apropriado para manter o jogo no caminho certo. Para coleções de agentes, o projeto de mecanismo nos permite construir sistemas inteligentes a partir de uma coleção de sistemas mais limitados — até mesmo sistemas não cooperativos —, quase do mesmo modo como as equipes de seres humanos conseguem atingir metas muito além do alcance de qualquer indivíduo.

Os exemplos de projeto de mecanismos incluem o leilão de passagens aéreas a baixo preço, o roteamento de pacotes TCP entre computadores, a decisão de como os médicos residentes serão designados para hospitais e a decisão de como os jogadores de futebol robóticos cooperarão com seus companheiros de equipes. O projeto de mecanismos se tornou mais que um assunto acadêmico na década de 1990, quando várias nações, diante do problema de leiloar licenças para transmissão pública em diversas bandas de frequência, perderam centenas de milhões de dólares em receita potencial como resultado de um projeto de mecanismo fraco. Formalmente, um **mecanismo** consiste em (1) uma linguagem para descrever o (possivelmente infinito) conjunto de estratégias permitidas que os agentes podem adotar, (2) um agente distinto, chamado **leiloeiro**, que reúne relatos de escolhas de estratégia dos agentes no jogo e (3) uma regra de resultado, conhecida por todos os agentes, que o leiloeiro usa para determinar as recompensas de cada agente, dadas as suas escolhas de estratégia.

17.6.1 Leilões

Vamos considerar primeiro os **leilões**. Um leilão é um mecanismo para vender alguns bens para membros de um grupo de consumidores. Para simplificar, nos concentraremos em leilões com um único item para venda.

Cada licitante i tem um valor de utilidade v_i para ter o item. Em alguns casos, cada agente ofertante tem um **valor privado** para o item. Por exemplo, o primeiro item vendido no eBay foi um ponteiro a *laser* quebrado, vendido por $14,83 para um colecionador de ponteiros a *laser* quebrados. Assim, sabemos que o colecionador tem $v_i \geq \$14{,}83$, mas a maioria das pessoas teria $v_j \ll \$14{,}83$. Em outros casos, como o leilão de direitos de perfuração para um tratado de petróleo, o item tem um **valor**

comum — o tratado vai produzir certa quantidade em dinheiro, *X*, e todos os ofertantes valem igualmente um dólar — mas há incerteza quanto ao valor real de *X*. Ofertantes diferentes têm informações diferentes e, portanto, estimativas diferentes do valor verdadeiro do item. Em ambos os casos, os ofertantes acabam com o seu próprio v_i. Dado o v_i, cada ofertante tem uma chance, no momento adequado ou nos momentos adequados no leilão, de fazer uma oferta b_i. O lance mais alto, b_{max}, ganha o item, mas o preço pago não precisa ser b_{max}; isso é parte do projeto de mecanismo.

O mecanismo de leilões mais conhecido é o **lance ascendente**,[8] ou **leilão inglês**, em que o leiloeiro começa pedindo um lance mínimo b_{min} (ou **reserva**). Se algum ofertante estiver disposto a pagar essa quantia, o leiloeiro, então, pede $b_{min} + d$, para algum incremento *d*, e continua a partir daí. O leilão termina quando ninguém estiver disposto a oferecer mais; então, o último ofertante ganha o item, pagando o preço do seu lance.

Como sabemos se esse é um bom mecanismo? Um dos objetivos é maximizar a receita esperada para o vendedor. Outro objetivo é maximizar uma noção de utilidade global. Essas metas se sobrepõem, até certo ponto, porque um aspecto da maximização da utilidade global é garantir que o vencedor do leilão seja o agente que mais valorizou o item (e, portanto, está disposto a pagar mais). Dizemos que um leilão é **eficiente** se os bens forem para o agente que mais os valoriza. O lance ascendente do leilão geralmente é eficiente e maximiza a receita, mas se o preço de reserva for muito alto o ofertante que mais o valoriza pode não concorrer e, se a reserva for muito baixa, o vendedor perde a receita líquida.

Provavelmente, uma das coisas mais importantes que um mecanismo de leilão pode fazer é encorajar um número suficiente de consumidores a entrar no jogo e desencorajá-los de se envolver em **coalisão**. Coalisão é um acordo injusto ou ilegal por dois ou mais licitantes para manipular os preços. Pode acontecer em acordos secretos de bastidores ou tacitamente, dentro das regras do mecanismo.

Por exemplo, em 1999, a Alemanha leiloou dez blocos de espectro de telefonia celular com um leilão simultâneo (eram consideradas apostas de todos os dez blocos, ao mesmo tempo), utilizando a regra de que qualquer lance deveria ter aumento mínimo de 10% sobre o lance do bloco anterior. Havia apenas dois ofertantes credíveis, e o primeiro, a Mannesman, entrou com a oferta de 20 milhões de marcos nos blocos 1 a 5 e 18,18 milhões nos blocos 6 a10. Por que 18,18M? Um dos gerentes da T-Mobile disse que eles “interpretaram o primeiro lance da Mannesman como uma oferta”. Ambas as partes podem calcular que um aumento de 10% sobre 18,18M é 19,99M; assim, o lance da Mannesman foi interpretado como se dissesse: “Cada um de nós pode obter a metade dos blocos por 20M; não vamos desperdiçá-la oferecendo preços mais elevados.” E, de fato, o lance da T-Mobile de 20M dos blocos 6-10 foi o fim do leilão. O governo alemão obteve menos do que o esperado porque os dois concorrentes foram capazes de usar o mecanismo de licitação para chegar a um acordo tácito sobre como não competir. Do ponto de vista do governo, poderia ter sido obtido resultado melhor por qualquer uma dessas alterações no mecanismo: preço de reserva mais alto; oferta selada de primeiro preço do leilão, para que os concorrentes não pudessem se comunicar através de seus lances; ou incentivo para trazer um terceiro licitante. Talvez a regra dos 10% tenha sido um erro no projeto de mecanismo porque facilitou a sinalização precisa da Mannesman para a T-Mobile.

Em geral, tanto o vendedor como a função utilidade global se beneficiam se houver mais interessados, embora a utilidade global possa sofrer se você contar o custo do tempo perdido dos ofertantes que não têm chance de ganhar. Uma maneira de encorajar mais os ofertantes é tornar o mecanismo mais fácil para eles. Afinal, ao requerer muita pesquisa ou cálculo por parte dos ofertantes, eles podem decidir empregar o seu dinheiro em outro lugar. Por isso, é desejável que os ofertantes tenham uma **estratégia dominante**. Lembre-se de que "dominante" significa que a estratégia funcione em oposição a todas as outras estratégias, que por sua vez significa que um agente pode adotá-la sem levar em conta as outras estratégias. Um agente com uma estratégia dominante pode simplesmente licitar, sem perder tempo contemplando possíveis estratégias de outros agentes. Um mecanismo pelo qual os agentes têm uma estratégia dominante é chamado de mecanismo a **prova de estratégia**. Se, como geralmente ocorre, essa estratégia envolve ofertantes revelando o seu valor verdadeiro, v_i, então é chamado de **leilão revelador da verdade** ou **confiável**; a expressão **incentivo compatível** também é usada. O princípio da revelação estabelece que qualquer mecanismo pode ser transformado em um mecanismo de revelação da verdade equivalente; então, parte do projeto do mecanismo é encontrar esses mecanismos equivalentes.

Acontece que o leilão de lance ascendente tem a maioria das propriedades desejáveis. O agente com o maior valor v_i recebe os bens ao preço de b_o + d, onde b_o é o lance mais alto entre todos os outros agentes e d é o incremento do leiloeiro.[9] Os ofertantes têm uma estratégia dominante simples: continuar oferecendo lances enquanto o custo atual for menor que seu v_i. O mecanismo não é exatamente revelador da verdade porque o ofertante vencedor revela apenas que seu $v_i \geq b_o + d$; temos um limite inferior em v_i, mas não uma quantidade exata.

Uma desvantagem (do ponto de vista do vendedor) do leilão de lance ascendente é que pode desencorajar a concorrência. Suponha que, em um lance para espectro de telefone celular, haja uma empresa favorecida e que todos concordam que seria capaz de alavancar os clientes existentes e a infraestrutura, podendo assim ter um lucro maior do que ninguém. Os concorrentes potenciais podem ver que não têm chance em um leilão de lance ascendente porque a empresa favorecida pode sempre dar um lance mais alto. Dessa forma, os competidores podem não entrar no leilão, e a empresa favorecida acaba ganhando ao preço de reserva.

Uma propriedade negativa do leilão inglês é seu alto custo de comunicação. Ou o leilão tem lugar em uma sala, ou todos os agentes precisam ter linhas de comunicação confiáveis de alta velocidade; em ambos os casos tem de haver tempo disponível para passar por várias rodadas de lances. Um mecanismo alternativo que exige bem menos comunicação é o **leilão de oferta lacrada**. Aqui, cada licitante faz uma única oferta e a comunica ao leiloeiro, sem os outros licitantes tomarem conhecimento dela. Com esse mecanismo, não há mais uma simples estratégia dominante. Se o seu valor é v_i e você acredita que a oferta máxima de todos os outros participantes será b_o, você deve ofertar $b_o + \varepsilon$, para algum ε pequeno, se for menos que v_i. Assim, o seu lance depende de sua estimativa dos lances de outros agentes, exigindo que você trabalhe mais. Observe, também, que o agente com o v_i maior pode não ganhar o leilão. Isso é compensado pelo fato de que o leilão é mais competitivo, reduzindo a propensão para um ofertante favorecido.

Uma pequena mudança no mecanismo de leilões de ofertas lacradas produz o **leilão de oferta lacrada de segundo preço**, também conhecido como **leilão de Vickrey**.[10] Em tais leilões, o

vencedor paga o preço da *segunda* oferta mais alta, b_o, em vez de pagar sua própria oferta. Essa simples modificação elimina completamente as deliberações complexas exigidas pelos leilões de ofertas lacradas padrão (ou **de primeiro preço**) porque agora a estratégia dominante é simplesmente oferecer v_i; o mecanismo é revelador da verdade. Observe que a utilidade do participante i em termos de sua oferta b_i, seu valor v_i e a melhor oferta entre os outros jogadores, b_o, é:

$$u_i = \begin{cases} (v_i - b_o) & \text{se } b_i > b_o \\ 0 & \text{em caso contrário.} \end{cases}$$

Para ver que $b_i = v_i$ é uma estratégia dominante observe que, quando $(v_i - b_o)$ é positivo, qualquer oferta que vença o leilão é ótima, e a oferta de v_i em particular vence o leilão. Por outro lado, quando $(v_i - b_o)$ é negativo, qualquer oferta que perde o leilão é ótima, e a oferta de v_i em particular perde o leilão. Assim, a oferta de v_i é ótima para todos os valores possíveis de b_o e, de fato, v_i é a única oferta que tem essa propriedade. Devido à sua simplicidade e aos requisitos mínimos de computação, tanto para o vendedor quanto para os ofertantes, o leilão de Vickrey é amplamente utilizado na construção de sistemas distribuídos de IA. Além disso, os mecanismos de busca na internet conduzem mais de um bilhão de leilões por dia para vender anúncios juntamente com os seus resultados de busca, e sites de leilão on-line manipulam $100 bilhões por ano em bens, todos utilizando variantes do leilão de Vickrey. Observe que o valor esperado para o vendedor é b_o, que é o mesmo retorno esperado que o limite do leilão inglês à medida que o incremento d tende a zero. Esse é realmente um resultado muito geral: o **teorema da equivalência de receitas**, com algumas ressalvas menores, afirma que qualquer mecanismo de leilão em que os licitantes com risco neutro têm valores v_i que apenas eles conhecem (mas conhecem a distribuição de probabilidade de onde esses valores são amostrados), vai produzir a mesma receita esperada. Esse princípio significa que os vários mecanismos não são concorrentes com base na geração de receitas, mas em outras qualidades.

Embora o leilão de segundo preço seja revelador da verdade, verifica-se que estender a ideia de múltiplos bens e utilizar um leilão de próximo preço não é revelador da verdade. Muitos mecanismos de busca na internet utilizam um mecanismo que oferece em leilão k janelas para comerciais em uma página. O maior lance ganha o primeiro lugar, o segundo fica em segundo, e assim por diante. Cada vencedor paga o preço do lance do próximo ofertante com valor inferior, com o entendimento de que o pagamento é feito somente se o pesquisador realmente clicar no anúncio. As janelas na parte superior são consideradas mais valiosas porque são mais propensas a ser vistas e clicadas. Imagine que três ofertantes, b_1, b_2 e b_3, tenham avaliações por um clique de $v_1 = 200$, $v_2 = 180$, e $v_3 = 100$, e que $k = 2$ janelas estão disponíveis; sabe-se que a janela mais bem posicionada é clicada 5% do tempo, e a pior posicionada, 2%. Se todos os ofertantes fizerem lances confiáveis, b_1 ganhará a janela superior e pagará 180, com retorno esperado de $(200 - 180) \times 0{,}05 = 1$. A segunda janela irá para b_2. Mas b_1 pode constatar que, se ofertasse qualquer coisa no intervalo entre 101-179, concederia a janela superior para b_2 ele ganharia a segunda janela e produziria um retorno esperado de $(200 - 100) \times 0{,}02 = 2$. Assim, nesse caso, b_1 pode dobrar o seu retorno esperado ao dar um lance menor do que o seu verdadeiro valor. Em geral, os ofertantes nesse leilão de múltiplas janelas

têm de gastar muita energia analisando as ofertas dos outros para determinar a sua melhor estratégia; não há estratégia simples dominante. Aggarwal *et al.* (2006) mostram que há um único mecanismo de leilão confiável para esse problema, em que o vencedor da janela j paga o preço total pela janela j apenas por aqueles cliques adicionais que estão disponíveis na janela j e não na janela $j + 1$. O vencedor paga o preço mais baixo pela janela para os cliques restantes. No nosso exemplo, b_1 iria ofertar 200 realmente, e pagaria 180 por 0,05 – 0, 02 = 0,03 cliques adicionais na janela superior, mas iria pagar apenas o custo da janela inferior, 100, pelos restantes 0,02 cliques. Assim, o retorno total para b_1 seria (200 − 180) × 0,03 + (200 − 100) × 0,02 = 2,6.

Outro exemplo em que os leilões podem ser úteis em IA é quando uma coleção de agentes está decidindo sobre cooperação em um plano conjunto. Hunsberger e Grosz (2000) mostraram que isso pode ser feito de forma eficiente com um leilão em que os agentes fazem lances por papéis no plano conjunto.

17.6.2 Bens comunitários

Agora vamos considerar outro tipo de jogo, em que os países estabelecem suas políticas de controle da poluição do ar. Cada país tem uma escolha: pode reduzir a poluição a um custo de −10 pontos para implementar as mudanças necessárias ou pode continuar a poluir, o que lhe confere uma utilidade de rede de −5 (em acréscimo a despesas de saúde etc.) e também contribui com −1 ponto para todos os outros países (porque o ar é compartilhado entre os países). É evidente que a estratégia dominante para cada país é “continuar a poluir”, mas se houver 100 países e cada um seguir essa política, cada país obterá uma utilidade total de −104, enquanto que se todo país reduzir a sua poluição cada um terá uma utilidade de −10. Essa situação é chamada de **tragédia das comunidades**: se ninguém tem de pagar para utilizar um recurso comum, ele tende a ser explorado de maneira que leva a baixar a utilidade total para todos os agentes. É semelhante ao dilema do prisioneiro: não há outra solução para o jogo que seja melhor para todas as partes, mas parece não haver maneira de os agentes racionais chegarem a essa solução.

A abordagem padrão para lidar com a tragédia das comunidades é mudar o mecanismo para aquele que impõe uma tarifa a cada agente pelo uso de bens comuns. Mais geralmente, precisamos garantir que todas as **externalidades** — efeitos sobre a utilidade global que não são reconhecidos nas transações dos agentes individuais — sejam explicitadas. A fixação dos preços corretamente é a parte difícil. No limite, essa abordagem equivale à criação de um mecanismo em que cada agente tenha de efetivamente maximizar a utilidade global, mas possa fazê-lo tomando uma decisão local. Para esse exemplo, um imposto sobre o carbono seria um exemplo de mecanismo que cobra pelo uso de um bem comum de uma forma que, se bem implementado, maximiza a utilidade global.

Como exemplo final, considere o problema de alocação de alguns bens comuns. Suponha que uma cidade decida que quer instalar alguns transceptores de internet sem fio gratuito. No entanto, o número de transceptores que a cidade pode adquirir é inferior ao número de bairros que o desejam. A cidade quer alocar os bens de forma eficiente aos bairros que mais os valorizariam. Ou seja, eles querem maximizar a utilidade global $V = \sum_i v_i$. O problema é que, se simplesmente for perguntado a cada conselho de bairro “o quanto você valoriza esse donativo gratuito?”, todos eles terão um

incentivo para mentir e relatar um valor mais alto. Acontece que existe um mecanismo, conhecido como **Vickrey-Clarke-Groves**, ou **VCG**, que torna isso uma estratégia dominante para cada agente ou relata a sua verdadeira utilidade, o que alcança uma alocação eficiente dos bens. O truque é que cada agente paga um imposto equivalente à perda da utilidade global que ocorre devido à presença do agente no jogo. O mecanismo funciona assim:

1. O leiloeiro pede a cada agente para relatar o seu valor para receber um item. Denominaremos b_i.
2. O leiloeiro destina os bens a um subconjunto de ofertantes. Chamaremos esse subconjunto de A e usaremos a notação $b_i(A)$ para significar o resultado de i sob essa atribuição: b_i, se i está em A (isto é, i é um vencedor) e 0 caso contrário. O leiloeiro escolhe A para maximizar a utilidade total relatada $B = \sum_i b_i(A)$.
3. O leiloeiro calcula (para cada i) a soma das utilidades relatadas para todos os vencedores, exceto i. Usamos a notação $B_{-i} = \sum_{j \neq i} b_j(A)$. O leiloeiro também calcula (para cada i) a alocação que maximizaria a utilidade global se i não estivesse no jogo; chamaremos essa soma de W_{-i}.
4. Cada agente i paga um imposto igual a $W_{-i} - B_{-i}$.

Nesse exemplo, a regra VCG significa que cada vencedor pagaria um imposto igual ao valor mais alto relatado entre os perdedores. Ou seja, se eu relatar que meu valor é 5 e isso fizer com que alguém com o valor 2 seja excluído de uma alocação, pagarei um imposto de 2. Todos os vencedores deverão ficar felizes por pagar um imposto inferior a seu valor, e todos os perdedores ficarão o mais feliz possível porque valorizam menos os bens do que o imposto exigido.

Por que esse mecanismo é revelador da verdade? Primeiro, considere o pagamento ao agente i, que é o valor da obtenção de um item menos o imposto:

$$v_i(A) - (W_{-i} - B_{-i}) \,. \tag{17.14}$$

Aqui distinguimos a verdadeira utilidade do agente, v_i, de sua utilidade relatada b_i (mas estamos tentando mostrar que a estratégia dominante é $b_i = v_i$). O agente i sabe que o leiloeiro vai maximizar a utilidade global usando os valores relatados,

$$\sum_j b_j(A) = b_i(A) + \sum_{j \neq i} b_j(A)$$

enquanto o agente i quer que o leiloeiro maximize (17.14), que pode ser reescrita como

$$v_i(A) + \sum_{j \neq i} b_j(A) - W_{-i} \,.$$

Desde que o agente i não possa afetar o valor da W_{-i} (que depende apenas de outros agentes), a única maneira como i pode fazer com que o leiloeiro otimize o que i quer é relatar a utilidade verdadeira, $b_i = v_i$.

17.7 RESUMO

Este capítulo mostrou como usar o conhecimento sobre o mundo para tomar decisões, mesmo quando os resultados de uma ação são incertos e as recompensas por uma ação podem não ser recebidas até muitas ações terem passado. Os principais pontos são:

- Problemas de decisão sequencial em ambientes incertos, também chamados **processos de decisão de Markov**, ou MDPs, são definidos por um **modelo de transição** que especifica os resultados probabilísticos de ações e uma **função de recompensa** que especifica a recompensa em cada estado.
- A utilidade de uma sequência de estados é a soma de todas as recompensas sobre a sequência, possivelmente descontadas com o passar do tempo. A solução de um MDP é uma **política** que associa uma decisão com todo estado que o agente possa alcançar. Uma política ótima maximiza a utilidade das sequências de estados encontradas quando ela é executada.
- A utilidade de um estado é a utilidade esperada das sequências de estados encontradas quando uma política ótima é executada, começando nesse estado. O algoritmo de **iteração de valor** para resolver MDPs funciona resolvendo iterativamente as equações que relacionam as utilidades de cada estado às de seus vizinhos.
- A **iteração de política** se alterna entre o cálculo das utilidades de estados sob a política atual e o aperfeiçoamento da política atual com relação às utilidades atuais.
- MDPs parcialmente observáveis, ou POMDPs, são muito mais difíceis de resolver que MDPs. Eles podem ser resolvidos pela conversão para um MDP no espaço contínuo de estados de crença; tanto os algoritmos de iteração de valor como de iteração de políticas foram delineados. O comportamento ótimo em POMDPs inclui a coleta de informações para reduzir a incerteza e, portanto, tomar melhores decisões no futuro.
- Um agente de teoria da decisão pode ser construído para ambientes de POMDPs. O agente utiliza uma **rede de decisão dinâmica** para representar os modelos de transição e observação, atualizar seu estado de crença e projetar para a frente possíveis sequências de ações.
- A **teoria dos jogos** descreve o comportamento racional para agentes em situações nas quais vários agentes interagem simultaneamente. As soluções de jogos são **equilíbrios de Nash** — perfis de estratégias em que nenhum agente tem incentivo para divergir da estratégia especificada.
- O **projeto de mecanismos** pode ser usado para definir as regras pelas quais os agentes vão interagir, a fim de maximizar alguma utilidade global pela operação de agentes individualmente racionais. Às vezes, existem mecanismos que atingem essa meta sem exigir que cada agente considere as escolhas feitas por outros agentes.

Retornaremos ao mundo de MDPs e POMDPs no Capítulo 21, quando estudarmos métodos de **aprendizado por reforço** que permitem a um agente melhorar seu comportamento a partir da experiência em ambientes sequenciais incertos.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Richard Bellman desenvolveu as ideias subjacentes à abordagem moderna para problemas de decisão sequencial ao trabalhar na RAND Corporation no início de 1949. Segundo sua autobiografia (Bellman, 1984), ele cunhou a incrível expressão "programação dinâmica" para ocultar do secretário da Defesa, Charles Wilson, que tinha fobia a pesquisa, o fato de que seu grupo estava trabalhando em matemática (isso não deve ser verdade realmente porque o seu primeiro trabalho utilizando a expressão (Bellman, 1952) apareceu antes de Wilson tornar-se secretário da Defesa, em 1953). O livro de Bellman, *Dynamic Programming* (1957), deu uma base sólida ao novo campo e introduziu as abordagens algorítmicas básicas. A tese de doutorado de Ron Howard (1960) introduziu a iteração de política e a ideia de recompensa média para resolver problemas de horizonte infinito. Vários resultados adicionais foram introduzidos por Bellman e Dreyfus (1962). A iteração de política modificada se deve a Van Nunen (1976) e a Puterman e Shin (1978). A iteração de política assíncrona foi analisada por Williams e Baird (1993), que também provaram o limite de perda de política da Equação 17.9. A análise de desconto em termos de preferências estacionárias se deve a Koopmans (1972). Os textos de Bertsekas (1987), Puterman (1994) e Bertsekas e Tsitsiklis (1996) fornecem uma introdução rigorosa a problemas de decisão sequencial. Papadimitriou e Tsitsiklis (1987) descrevem resultados sobre a complexidade computacional de MDPs.

O importante trabalho de Sutton (1988) e Watkins (1989) em métodos de aprendizado por reforço para resolução de MDPs desempenhou uma função significativa na introdução de MDPs na comunidade de IA, assim como a pesquisa posterior realizada por Barto *et al*. (1995) — anteriormente, o trabalho de Werbos (1977) continha muitas ideias semelhantes, mas não obteve tanta repercussão. A conexão entre MDPs e problemas de planejamento de IA foi realizada primeiro por Sven Koenig (1991), que mostrou que operadores probabilísticos de STRIPS fornecem uma representação compacta para modelos de transição — veja também Wellman (1990b). O trabalho de Dean *et al*. (1993) e de Tash e Russell (1994) tentou superar a análise combinatória de grandes espaços de estados usando um horizonte de busca limitado e estados abstratos. As heurísticas baseadas no valor de informações podem ser usadas para selecionar áreas do espaço de estados em que uma expansão local do horizonte resultará em aperfeiçoamento significativo na qualidade da decisão. Os agentes que utilizam essa abordagem podem ajustar seu esforço para lidar com a pressão do tempo e gerar alguns comportamentos interessantes, como a utilização de "caminhos batidos" familiares para descobrir rapidamente sua rota pelo espaço de estados sem ter de recalcular decisões ótimas em cada ponto.

Como se poderia esperar, os pesquisadores de IA levaram MDPs na direção de representações mais expressivas, que podem acomodar problemas muito maiores do que as representações atômicas tradicionais com base em matrizes de transição. O uso de uma rede bayesiana dinâmica para representar os modelos de transição foi uma ideia óbvia, mas o trabalho em **MDPs fatorados** (Boutilier *et al*., 2000;. Koller e Parr, 2000; Guestrin *et al*., 2003b) estendeu a ideia de representações estruturadas da função valor com melhorias demonstráveis em termos de complexidade.

MDPs relacionais (Boutilier *et al*., 2001; Guestrin *et al*., 2003a) vão um passo além, utilizando representações estruturadas para lidar com domínios com muitos objetos relacionados.

A observação de que um MDP parcialmente observável pode ser transformado em um MDP

regular com a utilização de estados de crença se deve a Astrom (1965) e Aoki (1965). O primeiro algoritmo completo para a solução exata de POMDPs — essencialmente o algoritmo de iteração de valor apresentado neste capítulo — foi proposto por Edward Sondik (1971) em sua tese de doutorado. — um artigo publicado mais tarde por Smallwood e Sondik (1973) contém alguns erros, embora seja mais acessível. Lovejoy (1991) participou dos primeiros 25 anos de pesquisa em POMDP, atingindo conclusões um pouco pessimistas sobre a viabilidade de solução para problemas de grande porte. A primeira contribuição importante dentro de IA foi o algoritmo Witness (Cassandra *et al*., 1994;. Kaelbling *et al*,.1998), uma versão melhorada da iteração de valor do POMDP. Logo se seguiram outros algoritmos, incluindo uma abordagem criada por Hansen (1998) que constrói uma política de modo incremental, sob a forma de um autômato de estados finitos. Um trabalho mais recente em IA focou métodos de iteração de valor **baseados em ponto**, que, a cada iteração, geram planos condicionais e a-vetores para um conjunto finito de estados de crença, em vez de para o espaço de crença inteiro. Lovejoy (1991) propôs tal algoritmo para uma grade fixa de pontos, uma abordagem adotada também por Bonet (2002). Um artigo influente por Pineau *et al.* (2003) sugeriu a geração de pontos alcançáveis através da simulação de trajetórias de uma forma um pouco gananciosa; Spaan e Vlassis (2005) observaram que é necessário gerar planos para apenas um subconjunto pequeno de pontos, selecionados aleatoriamente para melhorar os planos de iteração anteriores para todos os pontos no conjunto. Métodos atuais baseados em pontos — tal como iteração de política baseada em ponto (Ji *et al*., 2007) — podem gerar soluções quase ótimas para POMDP com milhares de estados. Como POMDPs são PSPACE-difícil (Papadimitriou e Tsitsiklis, 1987), novos progressos poderão exigir aproveitar vários tipos de estrutura dentro de uma representação fatorada.

A abordagem on-line — utilizando busca de observação antecipada para selecionar uma ação para o estado de crença atual — primeiro foi analisada por Satia e Lave (1973). O uso da amostragem em nós de acaso foi explorado analiticamente por Kearns *et al*. (2000) e Ng e Jordan (2000). As ideias básicas para uma arquitetura de agentes usando redes de decisão dinâmicas foram propostas por Dean e Kanazawa (1989a). O livro *Planning and Control* de Dean e Wellman (1991) trata o assunto com muito maior profundidade, fazendo conexões entre modelos de DBN/DDN e a literatura clássica de controle sobre filtragem. Tatman e Shachter (1990) mostraram como aplicar algoritmos de programação dinâmica a modelos de DDN. Russell (1998) explica várias maneiras pelas quais tais agentes podem ter sua escala aumentada e identifica várias questões abertas de pesquisa.

As raízes mais antigas da teoria dos jogos podem ser localizadas em propostas feitas no século XVII por Christiaan Huygens e Gottfried Leibniz para estudar interações humanas competitivas e cooperativas de forma científica e matemática. Ao longo do século XIX, vários economistas importantes criaram exemplos matemáticos simples para analisar casos específicos de situações competitivas. Os primeiros resultados formais em teoria dos jogos se devem a Zermelo (1913) (que, no ano anterior, sugeriu uma forma de busca de minimax para jogos, embora ela estivesse incorreta). Emile Borel (1921) introduziu a noção de estratégia mista. John Von Neumann (1928) provou que todo jogo de duas pessoas e de soma zero tem um equilíbrio de maximin em estratégias mistas e um valor bem definido. A colaboração de Von Neumann com o economista Oskar Morgenstern levou à publicação em 1944 do volume *Theory of Games and Economic Behavior*, o livro definitivo sobre teoria dos jogos. A publicação do livro foi atrasada pela escassez de papel na época da guerra, até um membro da família Rockefeller subsidiar pessoalmente sua publicação.

Em 1950, com 21 anos de idade, John Nash publicou suas ideias relativas a equilíbrios em jogos gerais. Sua definição de uma solução de equilíbrio, embora se originasse do trabalho de Cournot (1838), ficou conhecida como equilíbrio de Nash. Depois de um longo atraso devido à esquizofrenia que ele sofreu de 1959 em diante, Nash recebeu o Prêmio Nobel de Economia em memória (juntamente com Reinhart Selten e John Harsanyi) em 1994. O equilíbrio de Bayes-Nash é descrito por Harsanyi (1967) e discutido por Kadane e Larkey (1982). Algumas questões relacionadas ao uso da teoria dos jogos para controle de agentes são abordadas por Binmore (1982).

O dilema do prisioneiro foi criado como exercício de sala de aula por Albert W. Tucker em 1950 (baseado em um exemplo de Merrill Flood e Melvin Dresher) e foi amplamente focalizado por Axelrod (1985) e Poundstone (1993). Jogos repetidos foram introduzidos por Luce e Raiffa (1957), bem como jogos de informações parciais por Kuhn (1953). O primeiro algoritmo prático para jogos de informações parciais, sequenciais, foi desenvolvido dentro da IA por Koller *et al*. (1996); o artigo de Koller e Pfeffer (1997) fornece uma introdução legível à área em geral e descreve um sistema funcional para representar e resolver jogos sequenciais.

Billings et al. (2003) discutem o uso de abstração para reduzir uma árvore de jogo para um tamanho que possa ser resolvido com a técnica de Koller. Bowling *et al.* (2008) mostram como usar amostragem importante para obter melhor estimativa do valor de uma estratégia. Waugh *et al.* (2009) mostram que a abordagem da abstração é vulnerável para cometer erros sistemáticos na aproximação da solução de equilíbrio, o que significa que toda a abordagem está em um terreno de areia movediça: funciona para alguns jogos, mas não para outros. Korb *et al*. (1999) fizeram experimento com um modelo de oponente na forma de rede bayesiana. Ele joga com cinco cartas tão bem quanto os seres humanos experientes. Zinkevich *et al*. (2008) mostram como uma abordagem que minimiza o arrependimento pode encontrar o equilíbrio aproximado para abstrações com 10^{12} estados, 100 vezes mais do que os métodos anteriores.

A teoria dos jogos e os MDPs são combinados com os jogos da teoria de jogos de Markov, também chamados jogos estocásticos (Littman, 1994; Hu e Wellman, 1998). Shapley (1953) descreveu realmente o algoritmo de iteração de valor independentemente de Bellman, mas seus resultados não foram apreciados amplamente, talvez porque foram apresentados no contexto dos jogos de Markov. A teoria evolucionária dos jogos (Smith, 1982; Weibull, 1995) analisa a estratégia derivada ao longo do tempo: se a estratégia do adversário estiver mudando, como se deve reagir? Livros sobre a teoria dos jogos a partir de um ponto de vista econômico incluem os de Myerson (1991), Fudenberg e Tirole (1991), Osborne (2004) e Osborne e Rubinstein (1994); Mailath e Samuelson (2006) concentraram-se em jogos repetidos. Da perspectiva de IA temos Nisan *et al*. (2007), Leyton-Browne Shoham (2008) e Shoham e Leyton-Brown (2009).

Em 2007, o Prêmio Nobel de Economia foi para Hurwicz, Maskin e Myerson “por estabelecerem as bases da teoria de projeto de mecanismos” (Hurwicz, 1973). A tragédia das comunidades, um problema motivante para o campo, foi apresentado por Hardin (1968). A revelação do princípio foi devida a Myerson (1986), e o teorema de receita de equivalência foi desenvolvido independentemente por Myerson (1981) e Riley e Samuelson (1981). Dois economistas, Milgrom (1997) e Klemperer (2002), escreveram sobre os leilões de espectro de bilhões de dólares em que estavam envolvidos.

O projeto de mecanismo é usado no planejamento multiagente (Hunsberger e Grosz, 2000; Stone *et*

al., 2009) e escalonamento (Rassenti *et al.*, 1982). Varian (1995) apresenta um breve panorama com ligações com a literatura de ciência da computação, e Rosenschein e Zlotkin (1994) apresentam um tratamento do tamanho de um livro com aplicações de IA distribuído. Trabalhos relacionados sobre IA distribuído também aparecem com outros nomes, incluindo inteligência coletiva (Tumer e Wolpert, 2000; Segaran, 2000) e controle baseado em mercado (Clearwater, 1996). Desde 2001 acontece a Trading Agents Competition (TAC), em que os agentes tentam extrair o melhor lucro em uma série de leilões (Wellman *et al.*, 2001; Arunachalam e Sadeh, 2005). Artigos sobre problemas computacionais em leilões sempre aparecem na ACM Conferences on Electronic Commerce.

EXERCÍCIOS

17.1 Para o mundo 4 × 3 mostrado na Figura 17.1, calcule que quadrados podem ser alcançados a partir de (1,1) pela sequência de ações [*Acima*, *Acima, Direita*, *Direita, Direita*] e com que probabilidades. Explique como essa computação está relacionada à tarefa de predição (veja a Seção 15.2.1) para um modelo oculto de Markov.

17.2 Selecione o membro específico do conjunto de políticas que sejam ótimas para R(s) > 0 como mostrado na Figura 17.2(b) e calcule a fração do tempo que o agente gasta em cada estado, no limite, se a política for executada para sempre (Dica: construa a matriz de probabilidade de transição de estado para estado que corresponde à política e consulte o exercício 15.2.)

17.3 Suponha que definamos a utilidade de uma sequência de estados como sendo a recompensa *máxima* obtida em qualquer estado na sequência. Mostre que essa função utilidade não resulta em preferências estacionárias entre as sequências de estados. Ainda é possível definir a função utilidade em estados tais que a tomada de decisões de UME resulte em comportamento ótimo?

17.4 Às vezes, os MDPs são formulados com uma função recompensa $R(s, a)$ que depende da ação tomada ou com uma função de recompensa $R(s, a, s')$ que também depende do estado resultante.

- **a.** Escreva as equações de Bellman para essas formulações.
- **b.** Mostre como um MDP com função de recompensa $R(s, a, s')$ pode ser transformado em um MDP diferente com função recompensa $R(s, a)$, tal que políticas ótimas no novo MDP correspondam exatamente às políticas ótimas no MDP original.
- **c.** Agora faça o mesmo para converter MDPs com $R(s, a)$ em MDPs com $R(s)$.

17.5 Para o ambiente mostrado na Figura 17.1, encontre todos os valores de limiar para $R(s)$, tais que a política ótima se altere quando o limiar for cruzado. Você precisará de um modo para calcular a política ótima e seu valor para $R(s)$ fixo. [*Sugestão*: Prove que o valor de qualquer política fixa varia linearmente com $R(s)$.]

17.6 A Equação 17.7 (Seção 17.2.3) afirma que o operador de Bellman é uma contração.

- **a.** Mostre que, para quaisquer funções f e g,

$$|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)| \ .$$

b. Escreva uma expressão para $|(BU_i - BU'_i)(s)|$ e aplique o resultado de (a) para completar a prova de que o operador de Bellman é uma contração.

17.7 Este exercício considera os MDPs de dois jogadores que correspondem a jogos de revezamento de soma zero, como os do Capítulo 5. Sejam os jogadores *A* e *B*, e seja *R*(*s*) a recompensa para o jogador *A* em *s* (a recompensa para *B* é sempre igual e oposta).

a. Seja $U_A(s)$ a utilidade do estado *s* quando é a vez de *A* executar um movimento em *s*, e seja $U_B(s)$ a utilidade do estado *s* quando é a vez de *B* fazer um movimento em *s*. Todas as recompensas e utilidades são calculadas a partir de ponto de vista de *A* (exatamente como na árvore de jogo de minimax). Escreva equações de Bellman que definam $U_A(s)$ e $U_B(s)$.

b. Explique como realizar a iteração de valor de dois jogadores com essas equações e defina um critério de parada apropriado.

c. Considere o jogo descrito na Figura 5.17. Desenhe o espaço de estados (e não a árvore de jogo), mostrando os movimentos feitos por *A* como linhas contínuas e os movimentos de *B* como linhas tracejadas. Marque cada estado com *R*(*s*). Você descobrirá que é útil organizar os estados (s_A, s_B) em uma grade bidimensional, usando s_A e s_B como "coordenadas".

d. Agora, aplique a iteração de valor de dois jogadores para resolver esse jogo e derive a política ótima.

17.8 Considere o mundo 3 × 3 mostrado na Figura 17.14(a). O modelo de transição é o mesmo que o 4 × 3 da Figura 17.1: em 80% do tempo o agente vai à direção que seleciona, o resto do tempo ele se move em ângulo reto para a direção pretendida.

r	-1	+10
-1	-1	-1
-1	-1	-1

(a)

+50	-1	-1	-1	...	-1	-1	-1	-1
Start				...				
-50	+1	+1	+1	...	+1	+1	+1	+1

(b)

Figura 17.14 (a) o mundo 3 × 3 do Exercício 17.8. A recompensa para cada estado está indicada. O quadrado superior direito é um estado terminal. (b) O mundo 101 × 3 do Exercício 17.9 (omitindo 93 colunas idênticas no meio). O estado inicial tem recompensa 0.

Implemente a iteração de valor para esse mundo para cada valor de *r* a seguir. Use recompensas descontadas com um fator de desconto de 0,99. Mostre a política obtida em cada caso. Explique intuitivamente por que o valor de *r* conduz a cada política.

a. $r = 100$

b. $r = -3$

c. $r = 0$

d. $r = +3$

17.9 Considere o mundo 101 × 3 mostrado na Figura 17.14(b). No estado inicial, o agente tem uma

escolha de duas ações determinísticas, *Acima* ou *Abaixo*, mas nos outros estados o agente tem uma ação determinística *Direita*. Assumindo uma função recompensa descontada, para quais valores de desconto γ o agente deveria escolher *Acima* e para quais *Abaixo*? Calcule a utilidade de cada ação como uma função de γ. (Note que esse exemplo simples reflete realmente muitas situações do mundo real em que um deve pesar o valor de uma ação imediata em relação às consequências contínuas de longo prazo, como a escolha de despejar poluentes em um lago.)

17.10 Considere um MDP não descontado que tem três estados, (1, 2, 3), com recompensas –1, –2, 0, respectivamente. O estado 3 é um estado terminal. Nos estados 1 e 2 existem duas ações possíveis: *a* e *b*. O modelo de transição é:

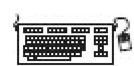
- No estado 1, a ação *a* move o agente para o estado 2 com probabilidade 0,8 e faz o agente ficar parado com probabilidade 0,2.
- No estado 2, a ação *a* move o agente para o estado 1 com probabilidade 0,8 e faz o agente ficar parado com probabilidade 0,2.
- No estado 1 ou no estado 2, a ação *b* move o agente para o estado 3 com probabilidade 0,1 e faz o agente ficar parado com probabilidade 0,9.

Responda às perguntas a seguir:

a. O que pode ser determinado *qualitativamente* sobre a política ótima nos estados 1 e 2?

b. Aplique a iteração de política mostrando completamente cada etapa, a fim de determinar a política ótima e os valores dos estados 1 e 2. Suponha que a política inicial tenha a ação *b* em ambos os estados.

c. O que acontecerá à iteração de política se a política inicial tiver a ação *a* em ambos os estados? O desconto ajudará? A política ótima dependerá do fator de desconto?

**17.11** Considere o mundo 4 × 3 mostrado na Figura 17.1.

a. Implemente um simulador de ambiente para esse ambiente, tal que a geografia específica do ambiente seja facilmente alterada. Algum código necessário para isso já está no repositório de código on-line.

b. Crie um agente que utilize iteração de política, e meça seu desempenho no simulador de ambiente a partir de vários estados iniciais. Execute vários experimentos a partir de cada estado inicial e compare a recompensa total média recebida por execução com a utilidade do estado, determinada por seu algoritmo.

c. Experimente aumentar o tamanho do ambiente. De que maneira o tempo de execução para a iteração de política varia com o tamanho do ambiente?

17.12 Como se pode usar o algoritmo de determinação de valor para calcular a perda esperada experimentada por um agente usando um determinado conjunto de estimativas de utilidade U e um modelo estimado P, em comparação com um agente utilizando valores corretos?

17.13 Seja b_0 o estado de crença inicial para que o POMDP 4 × 3 (Seção 17.4.1) seja a distribuição uniforme sobre os estados não terminais, ou seja, $\langle \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0, 0 \rangle$. Calcular o estado de crença exato b_1 após o agente executar *Esquerda* e seu sensor relatar uma parede adjacente. Calcule

também b_2 assumindo que o mesmo novamente acontecerá.

17.14 Qual é a complexidade de tempo das etapas de iteração do valor do POMDP para um ambiente sem sensores?

17.15 Considere uma versão do POMDP de dois estados no qual o sensor é 90% confiável no estado 0, mas não fornece nenhuma informação no estado 1 (isto é, relata 0 ou 1 com probabilidade igual). Analise, seja qualitativa ou quantitativamente, a função utilidade e a política ótima para esse problema.

17.16 Mostre que uma estratégia de equilíbrio dominante é um equilíbrio de Nash, e não *vice-versa*.

17.7 No jogo infantil pedra-papel-tesoura cada jogador revela ao mesmo tempo uma escolha de pedra, papel ou tesoura. O papel embrulha a pedra, a pedra deixa a tesoura cega e a tesoura corta o papel. Na versão extendida pedra-papel-tesoura-fogo-água, o fogo incide sobre a pedra, papel e tesoura; pedra, papel e tesoura batem na água; e a água incide sobre o fogo. Escreva a matriz de recompensa e encontre uma solução de estratégia mista para esse jogo.

17.8 A matriz de recompensa seguinte, de Blinder (1983), através de Bernstein (1996), mostra um jogo entre políticos e o Banco Federal.

	Fed: contrair	Fed:não fazer nada	Fed: expandir
Pol:contrair	F = 7, P = 1	F = 9, P = 4	F = 6, P = 6
Pol:não fazer nada	F = 8, P = 2	F = 5, P = 5	F = 4, P = 9
Pol: expandir	F = 3, P = 3	F = 2, P = 7	F = 1, P = 8

Os políticos podem expandir ou contrair a política fiscal, enquanto o Fed pode contrair ou expandir a política monetária. (E certamente os dois lados podem escolher não fazer nada.) Cada lado também tem preferências quanto a quem faz o que – nenhum dos lados deseja parcer ruim. As recompensas apresentadas são apenas as classificações das ordenações: 9 para a primeira escolha. Encontre o equilíbrio do jogo de Nash utilzando estratégia pura. Essa é uma solução ótima de Pareto? Talvez à essa luz você deseje analisar as políticas de administrações recentes.

17.19 Um leilão holandês é similar a um leilão inglês, mas, em vez de começar a licitação a um preço baixo e incremental, em um leilão holandês o vendedor começa com um preço elevado e, gradualmente, reduz o preço até que algum comprador esteja disposto a aceitar esse preço (se vários agentes aceitarem o preço, um será escolhido arbitrariamente como vencedor). Mais formalmente, o vendedor começa com um preço p e gradualmente reduz p em incrementos de d até que pelo menos um comprador aceite o preço. Assumindo que todos os ofertantes ajam racionalmente, é verdade que para um d arbitrariamente pequeno, um leilão holandês vai sempre resultar em um valor maior para oofertante obter o item? Se assim for, mostre matematicamente por quê. Se não, explique como pode ser possível que o ofertante com o maior valor para o item não o obtenha.

17.20 Imagine um mecanismo de leilão que é apenas como um leilão de lance ascendente, exceto que, ao final, o ofertante vencedor, aquele que oferta b_{max}, paga apenas $b_{max}/2$ em vez de b_{max}. Assumindo que todos os agentes são racionais, qual é a receita esperada para o leiloeiro para esse mecanismo,

em comparação com um leilão ascendente padrão?

17.21 Equipes da National Hockey League recebem historicamente 2 pontos se ganhar um jogo e 0 se perder. Se o jogo estiver empatado, haverá uma prorrogação; se ninguém ganhar na prorrogação, o jogo será um empate e cada equipe receberá um ponto. Mas a liga oficial sentiu que as equipes estavam jogando muito conservadoramente nas prorrogações (para evitar perda), e seria mais emocionante se a prorrogação produzisse um vencedor. Então, em 1999, os funcionários experimentaram um projeto de mecanismos: as regras foram alteradas, dando 1 ponto e não 0 para a equipe que perder na prorrogação. Continua 2 pontos por vitória e 1 para o empate.

a. O hóquei era um jogo de soma zero antes da mudança de regra? E depois?

b. Suponha que, em determinado período de tempo t em um jogo, o time da casa tenha probabilidade p de ganhar no tempo regulamentar, probabilidade $0{,}78 - p$ de perder e probabilidade 0,22 de entrar em prorrogação, na qual eles têm probabilidade θ de ganhar, $0{,}9 - \theta$ de perder e 0,1 de empatar. Represente as equações para o valor esperado para os times de casa e visitantes.

c. Imagine que seria legal e ético para as duas equipes fazerem um pacto pelo qual concordassem que iriam patinar para obter um empate no tempo regulamentar e, depois, ambas tentariam vencer seriamente na prorrogação. Em que condições, em termos de p e θ, seria racional para os dois times concordar com esse pacto?

d. Longley e Sankaran (2005) relataram que, desde que a regra mudou, o percentual de jogos com vencedor na prorrogação subiu 18,2%, conforme desejado, mas o percentual de prorrogações também subiu 3,6%. O que isso sugere sobre o possível jogo de coalisão ou conservador após a mudança na regra?

---

[1] Algumas definições de MDPs permitem que a recompensa dependa também da ação e do resultado e, assim, a função de recompensa é $R(s, a, s')$. Isso simplifica a descrição de alguns ambientes, mas não altera o problema em nenhum aspecto fundamental, como mostrado no Exercício 17.4.

[2] Embora isso pareça óbvio, não vale para as políticas de horizonte finito ou para outras formas de combinar recompensas ao longo do tempo. A prova decorre diretamente da singularidade da função utilidade sobre os estados, como mostrado na Seção 17.2.

[3] Tal como acontece com a função recompensa para MDPs, o modelo de sensoriamento também pode depender da ação e do estado resultante, mas novamente essa alteração não é fundamental.

[4] O jogo de par-ou-ímpar é uma versão recreativa de um **jogo de inspeção**. Em tais jogos, um inspetor escolhe um dia para inspecionar uma instalação (como um restaurante ou uma fábrica de armas biológicas), e o operador da instalação escolhe um dia para esconder todo o material irregular. O inspetor ganha se os dias forem diferentes, e o operador da instalação ganha se eles forem iguais.

[5] O caráter ótimo de Pareto tem esse nome em homenagem ao economista Vilfredo Pareto (1848-1923).

[6] Ou uma constante.

[7] É uma coincidência o fato de essas equações serem iguais às de p; a coincidência surge porque $U_E(um, dois) = U_E(dois, um) = -3$. Isso também explica por que a estratégia ótima é a mesma para ambos os jogadores.

[8] A palavra "*auction*" ("leilão" em português) vem do latim *augere*, "aumentar".

[9] Existe realmente uma pequena chance de que o jogador com v_i mais alto deixe de obter as mercadorias no caso em que $b_o < v_i < b_o + d$. A chance desse acontecimento pode ser arbitrariamente pequena diminuindo-se o incremento d.

[10] Esse nome é uma homenagem a William Vickrey (1914-1996), que ganhou o Prêmio Nobel de economia em 1996 por seu trabalho e

morreu de um ataque cardíaco três anos depois.

PARTE V

Aprendizagem

CAPÍTULO 18

Aprendendo a partir de exemplos

Em que descrevemos agentes que podem melhorar seu comportamento através do estudo diligente de suas próprias experiências.

Um agente estará **aprendendo** se melhorar o seu desempenho nas tarefas futuras de aprendizagem após fazer observações sobre o mundo. A aprendizagem pode variar do corriqueiro, como anotar um número de telefone, até o profundo, como mostrado por Albert Einstein, que inferiu uma nova teoria para o universo. Neste capítulo, vamos nos concentrar em uma classe de problema de aprendizagem, o que parece restrito, mas na verdade tem ampla aplicabilidade: a partir de uma coleção de pares de entrada e saída, aprender uma função que prevê a saída para novas entradas.

Por que iríamos querer um agente para aprender? Se o projeto do agente pode ser melhorado, para começar, por que os projetistas não apenas programam essa melhoria? Há três razões principais. Primeiro, os projetistas não podem antecipar todas as situações possíveis em que o agente possa se encontrar. Por exemplo, um robô projetado para navegar em labirintos tem de aprender a configuração de cada novo labirinto que encontra. Em segundo lugar, os projetistas não podem antecipar todas as mudanças ao longo do tempo; um programa projetado para prever os preços do mercado de ações de amanhã deve aprender a se adaptar quando as condições mudam do súbito crescimento ao fracasso. Terceiro, por vezes, os programadores humanos não têm ideia de como programar uma solução por si só. Por exemplo, a maioria das pessoas é boa em reconhecer o rosto dos membros da família, mas mesmo os melhores programadores são incapazes de programar um computador para realizar essa tarefa, exceto por meio de algoritmos de aprendizagem. Este capítulo primeiro dá uma visão geral das diferentes formas de aprendizagem, em seguida descreve uma abordagem popular, aprendizagem em árvore de decisão, na Seção 18.3, seguida por uma análise teórica da aprendizagem nas Seções 18.4 e 18.5. Verificamos vários sistemas de aprendizagem utilizados na prática: os modelos lineares, não lineares (em particular, redes neurais), não paramétricos e máquinas de vetores de suporte. Finalmente vamos mostrar como conjuntos de modelos podem superar um modelo único.

18.1 FORMAS DE APRENDIZAGEM

Qualquer componente de um agente pode ser melhorado através da aprendizagem a partir dos

dados. As melhorias e as técnicas usadas para construí-los depende de quatro fatores principais:

- Que *componente* deve ser melhorado.
- O *conhecimento prévio* que o agente já tem.
- Que *representação* é usada para os dados e para o componente.
- Que *feedback* está disponível para aprendizagem.

Componentes a serem aprendidos

O Capítulo 2 descreveu vários projetos de agentes. Os componentes desses agentes incluem:

1. Um mapeamento direto de condições no estado atual para ações.
2. Um meio para deduzir propriedades relevantes do mundo a partir da sequência de percepções.
3. Informações sobre o modo como o mundo evolui e sobre os resultados de ações possíveis que o agente pode executar.
4. Informações de utilidade indicando a desejabilidade de estados do mundo.
5. Informações de valores de ações indicando a desejabilidade de ações.
6. Metas que descrevem classes de estados cuja realização maximiza a utilidade do agente.

Cada um desses componentes pode ser aprendido a partir de realimentação apropriada. Por exemplo, considere o treinamento de um agente para se tornar motorista de táxi. Toda vez que o instrutor gritar “Freie!”, o agente poderá aprender uma regra de condição-ação sobre quando frear (componente 1); o agente também sabe toda vez que o instrutor não grita. Ao ver muitas imagens que lhe mostram ônibus, o agente pode aprender a reconhecê-los (2). Experimentando ações e observando os resultados — por exemplo, freando bruscamente em uma estrada molhada —, ele poderá aprender os efeitos de suas ações (3). Depois, se não receber nenhuma gorjeta de passageiros que foram sacudidos durante o percurso, poderá aprender um componente útil de sua função utilidade global (4).

Representação e conhecimento prévio

Vimos vários exemplos de representações para os componentes do agente: sentenças lógicas proposicionais e de primeira ordem para os componentes de um agente lógico; redes bayesianas para os componentes inferenciais de um agente de decisão teórica, e assim por diante. Os algoritmos de aprendizagem eficazes foram concebidos para todas essas representações. Este capítulo (e a maioria dos atuais de pesquisa em aprendizagem de máquina) abrange entradas que formam uma **representação fatorada** — um vetor de valores e atributos — e saídas que podem ser tanto um valor contínuo numérico como um valor discreto. O Capítulo 19 abrange funções e conhecimento prévio composto por sentenças lógicas de primeira ordem, e o Capítulo 20 concentra-se em redes bayesianas.

Há outra maneira de ver os vários tipos de aprendizagem. Dizemos que a aprendizagem de uma função geral ou regra (possivelmente incorreta) a partir de pares específicos de entrada-saída é chamada de **aprendizagem indutiva**. Veremos no Capítulo 19 que também podemos fazer **aprendizagem analítica** ou **dedutiva**: passar de uma regra geral conhecida a uma nova regra logicamente derivada porém útil, por permitir um processamento mais eficiente.

Feedback para aprender

Existem *três tipos de feedback* que determinam os três principais tipos de aprendizagem:

Na **aprendizagem não supervisionada**, o agente aprende padrões na entrada, embora não seja fornecido nenhum *feedback* explícito. A tarefa mais comum de aprendizagem não supervisionada é o **agrupamento**: a detecção de grupos de exemplos de entrada potencialmente úteis. Por exemplo, um agente de táxi pode desenvolver gradualmente um conceito de "dia de tráfego bom" e "dia de tráfego ruim" sem nunca ter sido rotulados exemplos de cada um deles por um professor.

Em **aprendizagem por reforço**, o agente aprende a partir de uma série de reforços — recompensas ou punições. Por exemplo, a falta de gorjeta ao final de uma corrida dá ao agente do táxi a indicação de que algo saiu errado. Os dois pontos de vitória no final de um jogo de xadrez informam ao agente que fez a coisa certa. Cabe ao agente decidir qual das ações anteriores ao reforço foram as maiores responsáveis por isso.

Na **aprendizagem supervisionada**, o agente observa alguns exemplos de pares de entrada e saída, e aprende uma função que faz o mapeamento da entrada para a saída. No componente 1 dos parágrafos anteriores, as entradas são percepções e a saída é fornecida por um instrutor que diz "Freie!" ou "Vire à esquerda". No componente 2, as entradas são imagens da câmera, e as saídas vêm de um instrutor que diz "isso é ônibus". Em 3, a teoria da frenagem é uma função de estados e ações de frenagem até a distância de parada. Nesse caso, o valor da saída está disponível diretamente da percepção do agente (após o fato); o ambiente é o instrutor.

Na prática, essas distinções nem sempre são tão nítidas. Na **aprendizagem semissupervisionada**, são dados alguns poucos exemplos rotulados e deve-se fazer o que puder de uma grande coleção de exemplos não rotulados. Mesmo os rótulos em si podem não ser as verdades oraculares que esperamos. Imagine que você esteja tentando construir um sistema para adivinhar a idade de uma pessoa a partir de uma foto. Você reúne alguns exemplos rotulados tirando fotos das pessoas e perguntando a idade. Isso é aprendizagem supervisionada. Mas, na realidade, algumas das pessoas mentiram sua idade. Não é só que haja ruído aleatório nos dados, mas as imprecisões são sistemáticas, e descobri-las é um problema de aprendizagem não supervisionada, envolvendo imagens, idades autorrelatadas e idades (desconhecidas) verdadeiras. Assim, tanto ruído como falta de rótulos cria um *continuum* entre aprendizagem supervisionada e não supervisionada.

18.2 APRENDIZAGEM SUPERVISIONADA

A tarefa de aprendizagem supervisionada é a seguinte:

Dado um **conjunto de treinamento** de N pares de exemplos de entrada e saída

$$(x_1, y_1), (x_2, y_2), \dots (x_n, y_n),$$

onde cada y_j foi gerado por uma função desconhecida $y = f(x)$,
descobrir uma função h que se aproxime da função verdadeira f.

Aqui x e y podem ter qualquer valor, não precisando ser números. A função h é uma **hipótese**.[1] Aprendizagem é uma busca através do espaço de hipóteses possíveis por aquele que terá um bom desempenho, mesmo em novos exemplos além do conjunto de treinamento. Para medir a precisão de uma hipótese, fornecemos um **conjunto de testes** de exemplos que são distintos do conjunto de treinamento. Dizemos que uma hipótese **generaliza** bem se prevê corretamente o valor de y para novos exemplos. Às vezes, a função f é estocástica — não é estritamente uma função de x, e o que temos de aprender é uma distribuição de probabilidade condicional, $\mathbf{P}(Y \mid x)$.

Quando a saída y for de um conjunto finito de valores (como *ensolarado, nublado* ou *chuvoso*), o problema da aprendizagem será chamado de **classificação**, e será chamado de classificação booleana ou binária se houver apenas dois valores. Quando y for um número (como temperatura de amanhã), o problema de aprendizagem é chamado de **regressão**. (Tecnicamente, a solução de um problema de regressão é encontrar uma expectativa condicional ou valor médio de y porque a probabilidade de acharmos *exatamente* o número de valor real certo para y é 0.)

A Figura 18.1 mostra um exemplo familiar: o ajuste de função de uma única variável a alguns pontos de dados. Os exemplos são pontos no plano (x, y), onde $y = f(x)$. Não sabemos qual é o f, mas vamos aproximá-lo com uma função h selecionada a partir de um **espaço de hipótese**, $\mathcal{H}$, que para este exemplo vamos tomar como conjunto de polinômios, tais como $x^5 + 3x^2 + 2$. A Figura 18.1(a) mostra alguns dados com um ajuste exato por linha reta (o polinômio $0{,}4x + 3$). A linha é chamada hipótese **consistente** porque concorda com todos os dados. A Figura 18.1(b) mostra um polinômio de grau alto que também é consistente com os mesmos dados. Isso ilustra o problema fundamental de aprendizagem indutiva: *como escolhemos entre várias hipóteses consistentes?* Uma resposta é preferir a hipótese consistente *mais simples* para os dados. Esse princípio é chamado de **navalha de Ockham**, devido ao filósofo inglês do século XIV, Guilherme de Ockham, que a usou para argumentar fortemente contra todos os tipos de complicações. A definição de simplicidade não é fácil, mas parece claro que um polinômio de grau 1 é mais simples do que um polinômio de grau 7 e, assim, (a) deve ser preferido a (b). Tornaremos essa intuição mais precisa na Seção 18.4.3.

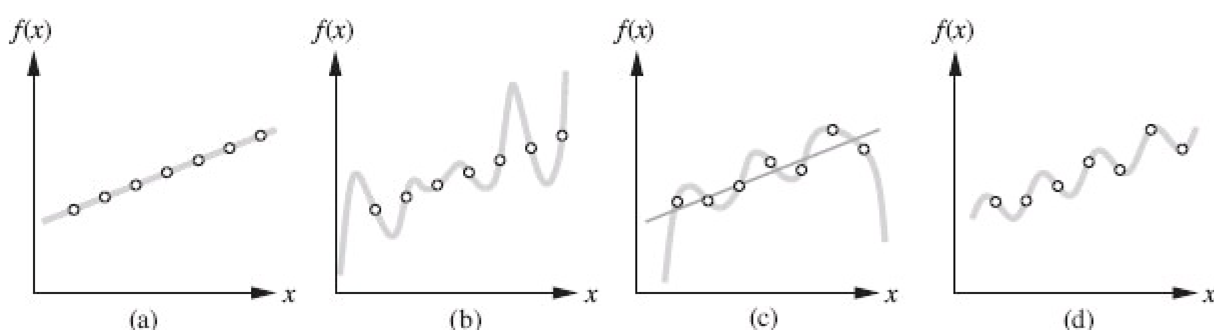


Figura 18.1 (a) Exemplo de pares $(x, f(x))$ e uma hipótese linear consistente. (b) Hipótese de polinômio de grau 7 consistente para o mesmo conjunto de dados. (c) Conjunto de dados diferente que admite um ajuste de polinômio de grau 6 exato ou um ajuste linear aproximado. (d) Um simples ajuste senoidal exato para o mesmo conjunto de dados.

A Figura 18.1(c) mostra um segundo conjunto de dados. Não existe nenhuma linha reta consistente para esse conjunto de dados; de fato, ele exige um polinômio de grau 6 (com sete parâmetros) para um ajuste exato. Existem apenas sete pontos de dados e, assim, o polinômio com sete parâmetros não parece encontrar qualquer padrão nos dados e não esperamos que ele generalize

bem. A linha reta que não é consistente com qualquer um dos pontos de dados, mas pode generalizar muito bem para os valores invisíveis de x, também é mostrada em (c). *Em geral, há uma compensação entre as hipóteses complexas que ajustam bem os dados de treinamento e as hipóteses mais simples que podem generalizar melhor*. Na Figura 18.1(d) expandimos o espaço de hipótese $\mathcal{H}$ para permitir polinômios sobre x e *seno*(x) e verificar que os dados em (c) podem ser ajustados exatamente por uma função simples da forma $ax + b + c$ seno(x). Isso mostra a importância da escolha do espaço de hipóteses. Dizemos que um problema de aprendizagem é **realizável** se o espaço de hipótese contiver a função verdadeira. Infelizmente, nem sempre podemos dizer se um problema de aprendizagem dado é realizável porque a função verdadeira não é conhecida.

Em alguns casos, um analista, ao verificar um problema, está querendo fazer distinções mais refinadas sobre o espaço de hipótese mesmo sem antes ter visto todos os dados, o que não significa apenas que uma hipótese é possível ou impossível, mas o quanto ela é provável. Pode-se fazer a aprendizagem supervisionada escolhendo a hipótese h^* que é mais provável, com os dados:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, P(h|dados)$$

Pela regra de Bayes isso é equivalente a

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, P(dados|h)\, P(h).$$

Então podemos dizer que a probabilidade prévia *P(h)* é alta para um polinômio de grau 1 ou 2, mas baixa para um polinômio de grau 7, e especialmente baixa para polinômios de grau 7 com grandes pontas acentuadas como na Figura 18.1(b). Permitimos funções de aparência incomum quando os dados informam que precisam delas realmente, mas a desencorajamos, dando-lhes uma probabilidade baixa *a priori*.

Por que não deixar $\mathcal{H}$ ser a classe de todos os programas em Java ou máquinas de Turing? Afinal, cada função computável pode ser representada por alguma máquina de Turing, e isso é o melhor que podemos fazer. Um problema com essa ideia é que ela não leva em conta a complexidade computacional da aprendizagem. *Há um compromisso entre a expressividade de um espaço de hipótese e a complexidade de encontrar uma boa hipótese dentro desse espaço.* Por exemplo, a adaptação de uma linha reta aos dados é um cálculo fácil, a adaptação de polinômios de grau alto é um pouco mais difícil, e a adaptação de máquinas de Turing em geral é indecidível. Uma segunda razão para preferir espaços de hipótese simples é que, presumivelmente, vamos querer usar *h* depois de termos aprendido isso, e é garantido que o cálculo de *h(x)* quando *h* for uma função linear é mais rápido, enquanto o cálculo em um programa arbitrário de máquina de Turing nem sequer se garante que termine. Por essas razões, a maioria dos trabalhos sobre aprendizagem tem se concentrado em representações simples.

Verificaremos que o compromisso entre expressividade e complexidade não é tão simples como parece à primeira vista: como vimos com lógica de primeira ordem, no Capítulo 8, uma linguagem expressiva torna possível que uma hipótese *simples* ajuste-se aos dados, enquanto restringir a expressividade da língua significa que qualquer hipótese consistente deve ser muito complexa. Por exemplo, as regras do xadrez podem ser escritas em uma página ou duas de lógica de primeira ordem, mas exigem milhares de páginas quando escritas em lógica proposicional.

18.3 APRENDIZAGEM EM ÁRVORES DE DECISÃO

A indução de árvores de decisão é uma das formas mais simples, e ainda assim mais bem-sucedidas, de aprendizagem de máquina. Primeiro, descreveremos a representação — o espaço de hipótese — e, em seguida, como aprender uma boa hipótese.

18.3.1 Representação da árvore de decisão

Uma **árvore de decisão** representa uma função que toma como entrada um vetor de valores de atributos e retorna uma "decisão" — um valor de saída único. Os valores de entrada e saída podem ser discretos ou contínuos. Por ora vamos nos concentrar em problemas em que a entrada tem valores discretos e a saída tem exatamente dois valores possíveis; isso é classificação booleana, em que cada exemplo é classificado como verdadeiro (**positivo**) ou falso (**negativo**).

Uma árvore de decisão alcança sua decisão executando uma sequência de testes. Cada nó interno na árvore corresponde a um teste do valor de um dos atributos de entrada, A_i, e as ramificações dos nós são classificadas com os valores possíveis do atributo, $A_i = v_{ik}$. Cada nó de folha na árvore especifica o valor a ser retornado pela função. A representação de árvores de decisão parece ser muito natural para os seres humanos; na realidade, muitos manuais do tipo "como fazer" (por exemplo, para consertos de automóveis) são inteiramente escritos como uma única árvore de decisão que se estende por centenas de páginas.

Como exemplo, construiremos uma árvore de decisão para decidir a espera ou não de uma mesa em um restaurante. Aqui, o objetivo é aprender uma definição para o **predicado de objetivo** *VaiEsperar*. Primeiro listaremos os atributos que vamos considerar como parte da entrada:

1. *Alternativa*: Se há um restaurante alternativo apropriado por perto.
2. *Bar*: Se o restaurante tem uma área de bar confortável onde se possa esperar.
3. *Sex/Sáb*: Verdadeiro às sextas e sábados.
4. *Faminto*: Se estamos com fome.
5. *Clientes*: Quantas pessoas estão no restaurante (os valores são: *Nenhum*, *Alguns* e *Cheio*).
6. *Preço*: A faixa de preços do restaurante ($, $$, $$$).
7. *Chovendo*: Se está chovendo do lado de fora.
8. *Reserva*: Se fizemos uma reserva.
9. *Tipo*: O tipo de restaurante (francês, italiano, tailandês ou só de hambúrguer).
10. *EsperaEstimada*: A espera estimada pelo gerente (0-10 minutos, 10-30, 30-60, >60).

Observe que cada variável tem um pequeno conjunto de valores possíveis, o valor de *EsperaEstimada*, por exemplo, não é um número inteiro, ao contrário, é um dos quatro valores discretos 0-10, 10-30, 30-60 ou >60. Na Figura 18.2 é mostrada a árvore de decisão geralmente usada por um de nós (SR) para esse domínio. Observe que a árvore ignora o *Preço* e o *Tipo* dos

atributos. Os exemplos são processados pela árvore a partir da raiz e seguindo a ramificação apropriada até alcançar uma folha. Por exemplo, uma situação com *Clientes* = *Cheio* e *EsperaEstimada* = 0-10 será classificada como positiva (isto é, sim, esperaremos por uma mesa).

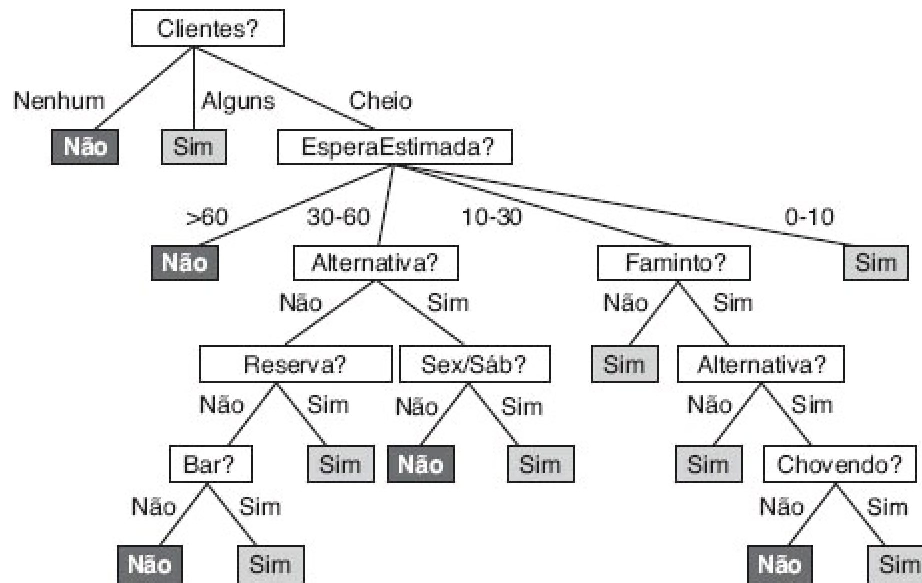


Figura 18.2 Árvore de decisão para definir se vamos ou não esperar por uma mesa.

18.3.2 Expressividade de árvores de decisão

Uma árvore de decisão booleana é logicamente equivalente à afirmação de que o atributo meta é verdadeiro se e somente se os atributos de entrada satisfizerem um dos caminhos que levam a uma folha com valor *verdadeiro*. Escrevendo isso na lógica proposicional, temos

$$Objetivo \Leftrightarrow (Caminho_1 \vee Caminho_2 \vee \ldots),$$

onde cada *Caminho* é uma conjunção de testes necessários de valores de atributos para seguir esse caminho. Assim, a expressão inteira é equivalente à forma normal disjuntiva, que significa que qualquer função na lógica proposicional pode ser expressa como uma árvore de decisão. Como exemplo, o caminho mais à direita na Figura 18.2 é

$$Caminho = (Clientes = Cheio \wedge EsperaEstimada = 0\text{-}10).$$

Para uma grande variedade de problemas, o formato da árvore de decisão gera um resultado agradável e conciso. Mas algumas funções não podem ser representadas de forma concisa. Por exemplo, a função da maioria, que retorna verdadeiro se e somente se mais da metade das entradas for verdadeira, exige uma árvore de decisão exponencialmente grande. Em outras palavras, as árvores de decisão são boas para alguns tipos de funções e ruins para outros. Existe *algum* tipo de representação que seja eficiente para todos os tipos de funções? Infelizmente, a resposta é não. Podemos mostrar isso de maneira geral. Considere o conjunto de todas as funções booleanas em n atributos. Quantas funções diferentes existem nesse conjunto? Esse é apenas o número de tabelas-verdade diferentes que podemos escrever porque a função é definida pela sua tabela-verdade. A

tabela-verdade sobre *n* atributos tem 2^n linhas, uma para cada combinação de valores dos atributos. Podemos considerar a coluna de “resposta” da tabela como o número com 2^n-bits que define a função. Isso significa que há 2^{2^n} funções diferentes (e haverá mais que esse número de árvores, já que mais de uma árvore pode calcular a mesma função). Isso é um número assustador. Por exemplo, com apenas 10 atributos booleanos do nosso problema do restaurante há $2^{1.024}$ ou cerca de 10^{308} funções diferentes para escolher, e, para 20 atributos, há cerca de $10^{300.000}$. Serão necessários alguns algoritmos engenhosos para encontrar boas hipóteses em tão grande espaço.

18.3.3 Indução de árvores de decisão a partir de exemplos

Um exemplo de árvore de decisão booleana consiste em um par $(\mathbf{x}, y)$, onde $\mathbf{x}$ é um vetor de valores para os atributos de entrada e *y* é um valor único de saída booleano. Um conjunto de treinamento de 12 exemplos são mostrados na Figura 18.3. Os exemplos positivos são aqueles em que a meta *VaiEsperar* é verdadeira ($\mathbf{x}_1$, $\mathbf{x}_3$, …); os exemplos negativos são aqueles em que ela é falsa ($\mathbf{x}_2$, $\mathbf{x}_5$, …).

Exemplo	Atributos										Meta
	Alt	*Bar*	*Sex*	*Fam*	*Cli*	*Preço*	*Chuva*	*Res*	*Tipo*	*Estim*	*VaiEsperar*
$\mathbf{x}_1$	*Sim*	*Não*	*Não*	*Sim*	*Alguns*	*$$$*	*Não*	*Sim*	*Francês*	*0-10*	$y_1 = Sim$
$\mathbf{x}_2$	*Sim*	*Não*	*Não*	*Sim*	*Cheio*	*$*	*Não*	*Não*	*Tailandês*	*30-60*	$y_2 = Não$
$\mathbf{x}_3$	*Não*	*Sim*	*Não*	*Não*	*Alguns*	*$*	*Não*	*Não*	*Hambúrguer*	*0-10*	$y_3 = Sim$
$\mathbf{x}_4$	*Sim*	*Não*	*Sim*	*Sim*	*Cheio*	*$*	*Sim*	*Não*	*Tailandês*	*10-30*	$y_4 = Sim$
$\mathbf{x}_5$	*Sim*	*Não*	*Sim*	*Não*	*Cheio*	*$$$*	*Não*	*Sim*	*Francês*	*>60*	$y_5 = Não$
$\mathbf{x}_6$	*Não*	*Sim*	*Não*	*Sim*	*Alguns*	*$$*	*Sim*	*Sim*	*Italiano*	*0-10*	$y_6 = Sim$
$\mathbf{x}_7$	*Não*	*Sim*	*Não*	*Não*	*Nenhum*	*$*	*Sim*	*Não*	*Hambúrguer*	*0-10*	$y_7 = Não$
$\mathbf{x}_8$	*Não*	*Não*	*Não*	*Sim*	*Alguns*	*$$*	*Sim*	*Sim*	*Tailandês*	*0-10*	$y_8 = Sim$
$\mathbf{x}_9$	*Não*	*Sim*	*Sim*	*Não*	*Cheio*	*$*	*Sim*	*Não*	*Hambúrguer*	*>60*	$y_9 = Não$
$\mathbf{x}_{10}$	*Sim*	*Sim*	*Sim*	*Sim*	*Cheio*	*$$$*	*Não*	*Sim*	*Italiano*	*10-30*	$y_{10} = Não$
$\mathbf{x}_{11}$	*Não*	*Não*	*Não*	*Não*	*Nenhum*	*$*	*Não*	*Não*	*Tailandês*	*0-10*	$y_{11} = Não$
$\mathbf{x}_{12}$	*Sim*	*Sim*	*Sim*	*Sim*	*Cheio*	*$*	*Não*	*Não*	*Hambúrguer*	*30-60*	$y_{12} = Sim$

Figura 18.3 Exemplos para o domínio de restaurante.

Queremos uma árvore que seja consistente com os exemplos e seja a menor possível. Infelizmente, não importa como medimos o tamanho, é um problema intratável encontrar a menor árvore consistente; não há maneira eficiente de busca através de 2^{2^n} árvores. Com uma heurística simples, no entanto, podemos encontrar uma boa solução aproximada: uma pequena (mas não a menor) árvore consistente. O algoritmo de APRENDIZAGEM EM ÁRVORE DE DECISÃO adota uma estratégia gulosa de dividir para conquistar: sempre testar o atributo mais importante em primeiro lugar. Esse

teste divide o problema em subproblemas menores que podem então ser resolvidos de forma recursiva. Por "atributo mais importante" referimo-nos àquele que faz mais diferença para a classificação de um exemplo. Dessa forma, esperamos obter a classificação correta, com um pequeno número de testes, o que significa que todos os caminhos na árvore serão curtos e a árvore como um todo será pouco profunda.

A Figura 18.4(a) mostra que *Tipo* é um atributo fraco porque nos deixa com quatro resultados possíveis, cada um dos quais tem o mesmo número de exemplos positivos e negativos. Por outro lado, na Figura 18.4(b), vemos que *Clientes* é um atributo bastante importante porque, se o valor é *Nenhum* ou *Alguns*, ficamos com conjuntos de exemplos para os quais podemos definitivamente responder (*Não* e *Sim*, respectivamente). Se o valor é *Cheio*, ficamos com um conjunto misto de exemplos. Em geral, depois que o primeiro teste de atributo separar os exemplos, cada resultado será um novo problema de aprendizagem de árvore de decisão em si, com menos exemplos e um atributo a menos. Existem quatro casos a considerar para esses problemas recursivos:

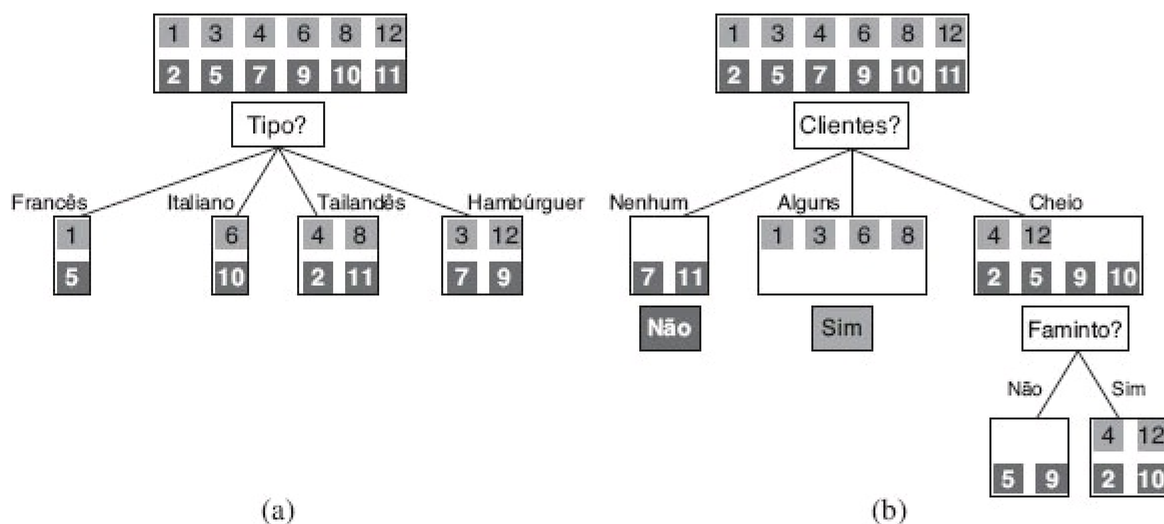


Figura 18.4 Divisão dos exemplos por meio de testes em atributos. A cada nó mostramos os exemplos remanescentes: positivos (caixas claras) e negativos (caixas escuras). (a) A divisão por *Tipo* não nos leva mais perto de distinguir entre exemplos positivos e negativos. (b) A divisão em Clientes faz um bom trabalho de separação de exemplos positivos e negativos. Após a divisão em *Clientes*, *Faminto* é um segundo teste razoavelmente bom.

1. Se todos os exemplos restantes forem positivos (ou todos negativos), terminamos: podemos responder *Sim* ou *Não*. A Figura 18.4(b) mostra exemplos disso nos casos *Nenhum* e *Alguns*.
2. Se existem alguns exemplos positivos e alguns negativos, escolha o melhor atributo para dividi-los. A Figura 18.4(b) mostra *Faminto* sendo usado para dividir os exemplos restantes.
3. Se não resta nenhum exemplo, isso significa que nenhum exemplo desse tipo foi observado, e retornamos um valor-padrão calculado a partir da classificação de maioria de todos os exemplos que foram utilizados na construção do nó pai. Esses são transmitidos na variável *exemplos-pais*.
4. Se não resta nenhum atributo mas há exemplos positivos e negativos, esses exemplos têm exatamente a mesma descrição, mas classificações diferentes. Isso acontece quando alguns dados estão incorretos (dizemos nesse caso que existe **ruído** nos dados) e também quando o domínio é não determinístico, ou ainda porque não podemos observar um atributo que

distinguiria os exemplos. O melhor que podemos fazer é voltar à classificação da maioria dos exemplos remanescentes.

O algoritmo de APRENDIZAGEM EM ÁRVORE DE DECISÃO é mostrado na Figura 18.5. Observe que o conjunto de exemplos é crucial para a *construção* da árvore, mas na árvore mesmo os exemplos não aparecem em nenhum lugar. Uma árvore é composta apenas de testes em atributos no interior de seu nós, valores de atributos nas ramificações e valores de saída nos nós folha. Os detalhes da função IMPORTÂNCIA são apresentados na Seção 18.3.4. A saída do algoritmo de aprendizagem na nossa amostra de conjunto de treinamento é mostrada na Figura 18.6. A árvore é claramente diferente da árvore original mostrada na Figura 18.2. Pode-se concluir que o algoritmo de aprendizagem não está fazendo um trabalho muito bom na aprendizagem da função correta. No entanto, isso seria a conclusão errada. O algoritmo de aprendizagem verifica os exemplos, e não a função correta e, de fato, sua hipótese (Figura 18.6) não só é compatível com todos os exemplos, mas é consideravelmente mais simples que a árvore original! O algoritmo de aprendizagem não tem razão para incluir testes para *Chuva e Reserva* porque pode classificar todos os exemplos sem eles. Foi também detectado um padrão interessante e previamente insuspeito: o primeiro autor vai esperar por comida tailandesa em fins de semana. Também se vê limitado a cometer erros em casos em que não haja exemplos. Por exemplo, ele nunca viu um caso em que a espera é 0-10 minutos, mas o restaurante está cheio.

função APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO(*exemplos, atributos, exemplos-pais)*
 retorna uma árvore de decisão
  se exemplos é vazio **então retornar** VALOR-DA-MAIORIA (*exemplos_pais*)
  senão se todos os *exemplos* têm a mesma classificação **então retornar** a classificação
  senão se *atributos* é vazio **então retornar** VALOR-DA-MAIORIA(*exemplos*)
  senão
    A ← argmax $_{a \in \text{atributes}}$IMPORTÂNCIA (*a*, *exemplos*)
    árvore ← uma nova árvore de decisão com teste de raiz *A*
    para cada valor v_k de *A* **faça**
      exs ← {*e* : *e* ∈ *exemplos* **e** *e.A* = v_k}
      subárvore ← APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO (*exs, atributos — A, exemplos*)
      adicionar uma ramificação à árvore com rótulo (*A* = v_k) e subárvore *subárvore*
    retornar *árvore*

Figura 18.5 O algoritmo de aprendizagem de árvore de decisão. A função IMPORTÂNCIA será descrita na Seção 18.3.4. A função VALOR-DA MAIORIA seleciona o valor de saída mais comum em um conjunto de exemplos, resolvendo os empates aleatoriamente.

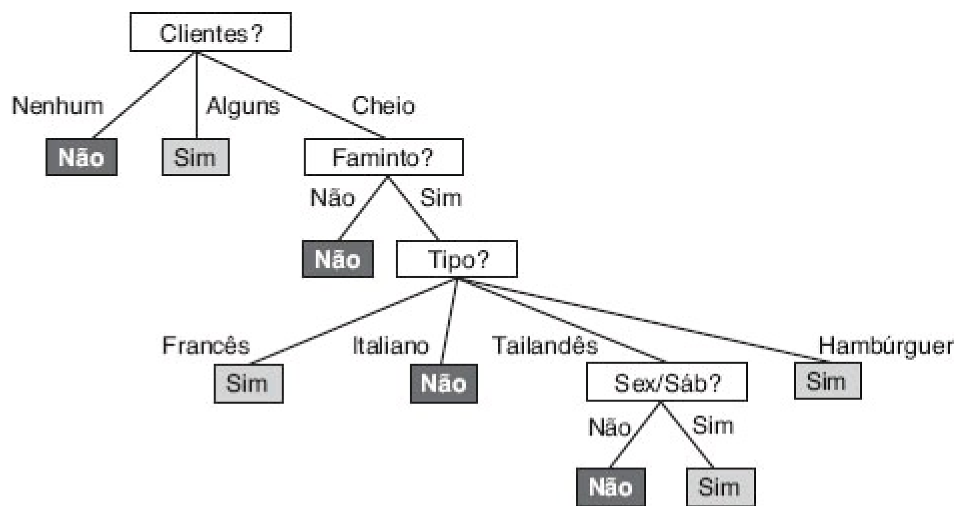


Figura 18.6 Árvore de decisão induzida a partir do conjunto de treinamento de 12 exemplos.

Nesse caso, não é para esperar quando *Faminto* for falso, mas eu (SR) certamente vou esperar. Com mais exemplos de treinamento, o programa de aprendizagem poderia corrigir esse erro.

Observamos que há um perigo de superinterpretar a árvore que o algoritmo seleciona. Quando existem diversas variáveis de importância similar, a escolha entre elas é um tanto arbitrária: com exemplos de entrada ligeiramente diferentes, em primeiro lugar, seria escolhida uma variável diferente para dividir, e a árvore toda pareceria completamente diferente. A função calculada pela árvore ainda seria semelhante, mas a estrutura da árvore poderia variar muito.

Pode-se avaliar a precisão de um algoritmo de aprendizagem com uma **curva de aprendizagem**, como mostrado na Figura 18.7. Temos 100 exemplos à nossa disposição, que dividimos em um conjunto de treinamento e em um conjunto de teste. Aprendemos uma hipótese h com o conjunto de treinamento e medimos a sua precisão com o conjunto de teste. Fazemos isso começando com um conjunto de treinamento de tamanho 1 e aumentando um de cada vez até o tamanho 99. Para cada tamanho realmente repetimos o processo de dividir 20 vezes aleatoriamente e tiramos a média dos resultados de 20 tentativas. A curva mostra que, à medida que o tamanho do conjunto de treinamento cresce, a precisão aumenta (por essa razão, as curvas de aprendizagem são também chamadas de **curvas felizes**). Nessa curva nós atingimos 95% de precisão, e parece que com mais dados a curva pode continuar a aumentar.

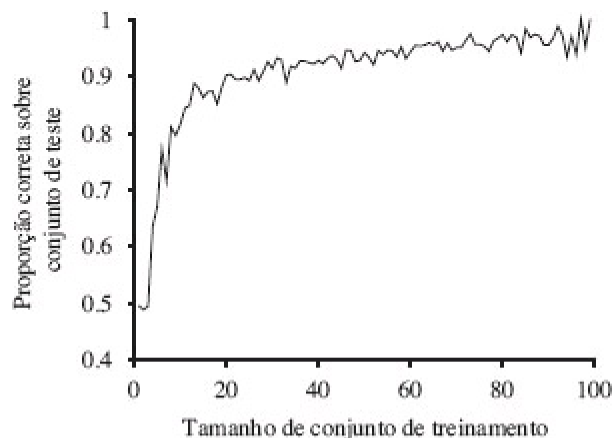


Figura 18.7 Curva de aprendizagem do algoritmo de aprendizagem em árvore de decisão em 100 exemplos gerados aleatoriamente no domínio do restaurante. Cada ponto de dados é a média de 20 tentativas.

18.3.4 Escolha de testes de atributos

A busca gulosa utilizada em aprendizagem de árvore de decisão foi projetada para minimizar aproximadamente a profundidade da árvore final. A ideia é escolher o atributo que vá o mais longe possível na tentativa de fornecer uma classificação exata dos exemplos. Um atributo perfeito divide os exemplos em conjuntos, cada um dos quais será todo positivo ou negativo que, então, se tornarão as folhas da árvore. O atributo *Clientes* não é perfeito, mas é bastante bom. Um atributo realmente inútil, como *Tipo*, deixa os conjuntos de exemplos com aproximadamente a mesma proporção de exemplos positivos e negativos do conjunto original.

Então, tudo de que precisamos é uma medida formal de "bastante bom" e "realmente inútil", e poderemos implementar a função IMPORTÂNCIA da Figura 18.5. Usaremos a noção de ganho de informação, que é definida em termos de **entropia**, a quantidade fundamental em teoria da informação (Shannon e Weaver, 1949).

A entropia é uma medida da incerteza de uma variável aleatória; a aquisição de informação corresponde a uma redução na entropia. A variável aleatória com um único valor — uma moeda que sempre dá cara — não tem incerteza e, portanto, sua entropia é definida como zero; assim, obtemos a informação observando o seu valor. O lançamento de uma moeda honesta é igualmente provável de dar cara ou coroa, 0 ou 1; mostraremos em breve que isso vale como "1 bit" de entropia. O lançamento de um dado honesto de quatro lados tem 2 bits de entropia porque toma dois bits para descrever uma de quatro escolhas igualmente prováveis. Agora, considere uma moeda viciada que dá cara 99% do tempo. Intuitivamente, essa moeda tem menos incerteza do que a moeda honesta — se apostarmos em cara estaremos errados apenas 1% do tempo — de modo que gostaríamos que ela tivesse uma medida de entropia perto de zero, mas positiva. Em geral, a entropia de uma variável aleatória V com valores v_k cada um com probabilidade $P(v_k)$, é definida como

$$\textit{Entropia:}\quad H(V) = \sum_k P(v_k)\log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k)\log_2 P(v_k)\,.$$

Podemos verificar que a entropia de um lançamento de uma moeda honesta é realmente de 1 bit:

$$H(\textit{honesta}) = -(0{,}5 \log_2 0{,}5 + 0{,}5 \log_2 0{,}5) = 1.$$

Se a moeda for adulterada para dar 99% cara, obtemos

$$H(\textit{adulterada}) = -(0{,}99 \log_2 0{,}99 + 0{,}01 \log_2 0{,}01) \approx 0{,}08 \text{ bits}.$$

Ela vai ajudar a definir $B(\theta)$ como a entropia de uma variável aleatória booleana que é verdadeira com probabilidade θ:

$$B(\theta) = -(\theta \log_2 q + (1-\theta) \log_2 (1-\theta)).$$

Assim, $\mathcal{H}(\textit{Adulterada}) = B(0{,}99) \approx 0{,}08$. Agora vamos voltar para a aprendizagem da árvore de decisão. Se um conjunto de treinamento contiver p exemplos positivos e n exemplos negativos, a

entropia do atributo meta em todo o conjunto será

$$H(Meta) = B\left(\frac{p}{p+n}\right).$$

O conjunto de treinamento do restaurante na Figura 18.3 tem $p = n = 6$, então a entropia correspondente é $B(0,5)$ ou exatamente 1 bit. Um teste em um único atributo A pode nos dar apenas uma parte desse bit 1. Podemos medir exatamente o quanto verificando a entropia remanescente *após* o teste do atributo.

Um atributo A com valores distintos d divide o conjunto de treinamento E em subconjuntos $E_1, \ldots, E_d$. Cada subconjunto E_k tem p_k exemplos positivos e n_k exemplos negativos; por isso, se continuarmos ao longo dessa ramificação, precisaremos de $B(p_k/(p_k + n_k))$ bits adicionais de informação para responder à pergunta. Um exemplo escolhido aleatoriamente do conjunto de treinamento tem o valor k-ésimo para o atributo com probabilidade $(p_k + n_k)/(p + n)$, então a entropia esperada remanescente após testar o atributo A é

$$Resto(A) = \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right).$$

O **ganho de informação** do teste do atributo em A é a redução esperada na entropia:

$$Ganho(A) = B\left(\frac{p}{p+n}\right) - Resto(A).$$

De fato $Ganho(A)$ é justamente o que precisamos para implementar a função IMPORTÂNCIA. Retornando aos atributos considerados na Figura 18.4, temos

$$Ganho(Clientes) = 1 - \left[\frac{2}{12}B\left(\frac{0}{2}\right) + \frac{4}{12}B\left(\frac{4}{4}\right) + \frac{6}{12}B\left(\frac{2}{6}\right)\right] \approx 0.541 \text{ bits},$$

$$Ganho(Tipo) = 1 - \left[\frac{2}{12}B\left(\frac{1}{2}\right) + \frac{2}{12}B\left(\frac{1}{2}\right) + \frac{4}{12}B\left(\frac{2}{4}\right) + \frac{4}{12}B\left(\frac{2}{4}\right)\right] = 0 \text{ bits},$$

confirmando a nossa intuição de que *Clientes* é um atributo melhor para dividir. Na verdade, *Clientes* tem o ganho máximo de qualquer dos atributos e seria escolhido pelo algoritmo de aprendizagem em árvore de decisão como a raiz.

18.3.5 Generalização e superadaptação

Em alguns problemas, o algoritmo APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO vai gerar uma grande árvore quando realmente não houver padrão a ser encontrado. Considere o problema de tentar prever se o lançamento de um dado vai dar 6 ou não. Suponha que os experimentos sejam realizados com dados diferentes e que os atributos que descrevem cada exemplo de treinamento incluem a cor do dado, seu peso, o tempo em que o lançamento foi feito e se os testadadores tinham os dedos cruzados. Se os dados forem honestos, a aprendizagem correta será uma árvore com um único nó que diz “não”, mas o algoritmo APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO vai se

aproveitar de qualquer padrão que puder encontrar na entrada. Se for descoberto que há dois lances de um dado azul de 7 g com os dedos cruzados e os dois dão 6, então o algoritmo pode construir um caminho que prevê 6 nesse caso. Esse problema é chamado de **superadaptação**. Um fenômeno geral, a superadaptação ocorre com todos os tipos de aprendizes, mesmo quando a função de destino não for de todo aleatória. Na Figura 18.1(b) e (c), vimos funções polinomiais sobreadaptando os dados. A superadaptação torna-se mais provável à medida que o espaço de hipótese e o número de atributos de entrada cresce, e menos provável à medida que aumentamos o número de exemplos de treinamento.

Para árvores de decisão, uma técnica chamada **poda de árvore de decisão** combate a superadaptação. A poda funciona através da eliminação dos nós que não são claramente relevantes. Começamos com uma árvore cheia, como a gerada pela APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO. Em seguida, verificamos um nó de teste que tem somente nós folha como descendentes. Se o teste parece ser irrelevante — detecta apenas o ruído dos dados —, eliminamos o teste, substituindo-o por um nó folha. Repetimos esse processo, considerando cada teste apenas com descendentes folha, até que cada um seja podado ou aceito como é.

A questão é como podemos detectar que um nó está testando um atributo irrelevante. Suponha que estejamos em um nó consistindo os exemplos em p positivo e n negativos. Se o atributo for irrelevante, seria de esperar que dividisse os exemplos em subconjuntos com aproximadamente a mesma proporção de exemplos positivos como o conjunto todo, $p/(p+n)$ e, assim, o ganho de informação estaria perto de zero.[2] Desse modo, o ganho de informações é uma boa pista para a irrelevância. Agora, a pergunta é: qual o tamanho do ganho que devemos exigir para fazer uma divisão baseada em um atributo específico?

Podemos responder a essa pergunta usando um **teste de significância** estatístico. Tal teste começa pela suposição de que não existe nenhum padrão subjacente (a hipótese conhecida como **hipótese nula**). Então, os dados reais são analisados para se calcular até que ponto eles divergem de uma ausência perfeita de padrão. Se o grau de desvio for estatisticamente improvável (em geral, adotado como a média para indicar probabilidade de 5% ou menos), ele será considerado como boa evidência da presença de um padrão significativo nos dados. As probabilidades são calculadas a partir de distribuições-padrão da proporção de desvio que se esperaria ver em uma amostragem aleatória.

Nesse caso, a hipótese nula é que o atributo é irrelevante e, consequentemente, que o ganho de informações para uma amostra infinitamente grande seria zero. Precisamos calcular a probabilidade de que, sob a hipótese nula, uma amostra de tamanho $v = n + p$ exiba o desvio observado a partir da distribuição esperada de exemplos positivos e negativos. Podemos medir o desvio comparando os números reais de exemplos positivos e negativos em cada subconjunto, p_k e n_k, com os números esperados, $\hat{p}_k$ e $\hat{n}_k$, supondo irrelevância verdadeira:

$$\hat{p}_k = p \times \frac{p_k + n_k}{p + n} \qquad \hat{n}_k = n \times \frac{p_k + n_k}{p + n} .$$

Uma medida conveniente do desvio total é dada por:

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k} .$$

Sob a hipótese nula, o valor de Δ é distribuído de acordo com a distribuição χ^2 (qui-quadrado) com $v - 1$ graus da liberdade. Podemos usar uma tabela χ^2 ou uma rotina de biblioteca padrão de estatística para ver se um valor Δ particular confirma ou rejeita a hipótese nula. Por exemplo, considere o atributo do tipo restaurante, com quatro valores e, portanto, três graus de liberdade. Um valor de $\Delta = 7{,}82$ ou mais rejeitaria a hipótese nula ao nível de 5% (e um valor de $\Delta = 11{,}35$ ou mais rejeitaria ao nível de 1%). O Exercício 18.8 pede para fazer as mudanças apropriadas em APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO para implementar essa forma de poda, que é conhecida como **poda** χ^2.

Com a poda, o ruído nos exemplos podem ser tolerados. Erros no rótulo do exemplo (por exemplo, um exemplo (**x**, *Sim*) que deve ser (**x**, *Não*)) fornece um aumento linear no erro de previsão, enquanto os erros nas descrições de exemplos (por exemplo, *Preço* = $ quando na verdade era *Preço* = $$) tem um efeito assintótico que piora à medida que a árvore encolhe para conjuntos menores. As árvores podadas desempenham significativamente melhor do que as árvores não podadas quando os dados contêm grande quantidade de ruído. Além disso, as árvores podadas geralmente são muito menores e, portanto, mais fáceis de entender.

Um aviso final: você poderia pensar que a poda χ^2 e o ganho de informação parecem semelhantes, então por que não combiná-los usando uma abordagem chamada **parada antecipada** — a propriedade do algoritmo da árvore de decisão parar de gerar nós quando não houver um bom atributo para dividir, em vez de enfrentar todo o problema de geração de nós para então podá-los? O problema com a parada antecipada é que impede de reconhecer situações em que não há um atributo bom, mas há combinações de atributos que são informativos. Por exemplo, considere a função XOR de dois atributos binários. Se houver um número aproximadamente igual de exemplos para todas as quatro combinações de valores de entrada, nenhum atributo será informativo, e o correto a fazer é dividir um dos atributos (não importa qual) e depois, no segundo nível, obteremos divisões que são informativas. A parada antecipada perderia isso, mas gerar e podar trataria isso corretamente.

18.3.6 Ampliando a aplicabilidade de árvores de decisão

Para estender a indução de árvore de decisão a uma variedade mais ampla de problemas, devemos atacar várias questões. Mencionaremos rapidamente várias, sugerindo que a compreensão total será obtida fazendo-se os exercícios associados:

- **Omissão de dados:** Em muitos domínios, nem todos os valores de atributos serão conhecidos para todo exemplo. Os valores podem não ter sido registrados ou talvez seja dispendioso demais obtê-los. Isso dá origem a dois problemas: primeiro, dada uma árvore de decisão completa, como se deve classificar um objeto em que esteja faltando um dos atributos de teste? Em segundo lugar, como se deve modificar a fórmula de ganho de informações quando alguns exemplos têm valores desconhecidos para o atributo? Essas perguntas serão tratadas no Exercício 18.9.

- **Atributos multivalorados**: Quando um atributo tem muitos valores possíveis, a medida de ganho de informações fornece uma indicação imprópria da utilidade do atributo. No caso extremo, um atributo como *ExactTime* tem valor diferente para cada exemplo, o que significa que cada subconjunto de exemplos é avulso com classificação única, e a medida do ganho de informação terá seu maior valor para esse atributo. Mas, em primeiro lugar, a escolha dessa divisão é pouco provável de produzir a melhor árvore. Uma solução é usar a **razão de ganho** (Exercício 18.10). Outra possibilidade é permitir que um teste booleano da forma $A = v_k$, isto é, tomar apenas um dos valores possíveis para um atributo, deixando os valores restantes para serem testados eventualmente depois na árvore.
- **Atributos de entrada com valores contínuos e inteiros**: Os atributos de valores contínuos ou inteiros, como *Altura* e *Peso*, têm um conjunto infinito de valores possíveis. Em vez de gerar um número infinito de ramificações, os algoritmos de aprendizagem de árvore de decisão em geral encontram o ponto de divisão que fornece o mais alto ganho de informações. Por exemplo, em dado nó na árvore, talvez a realização de testes sobre *Peso* >160 forneça o máximo de informações. Existem métodos eficientes para encontrar pontos de divisão bons: começar classificando os valores do atributo e depois considerar apenas os pontos de divisão que estão entre dois exemplos em ordem com diferentes classificações, enquanto acompanhando a execução dos totais de exemplos positivos e negativos de cada lado do ponto de divisão. A divisão é a parte mais cara das aplicações de aprendizagem em árvore de decisão do mundo real.
- **Atributos de saída com valores contínuos:** Se estamos tentando prever um valor numérico de saída, como o preço de um apartamento, precisamos de uma **árvore de regressão**, em vez de uma árvore de classificação. Uma árvore de regressão tem em cada folha uma função linear de um subconjunto de atributos numéricos, em vez de um único valor. Por exemplo, a ramificação para o apartamento de dois quartos pode acabar em uma função linear de metragem quadrada, número de banheiros e a renda média da vizinhança. O algoritmo de aprendizagem deve decidir quando interromper a divisão e começar a aplicar a regressão linear (consulte a Seção 18.6) sobre os atributos.

Um sistema de aprendizagem em árvore de decisão para aplicações reais deve ser capaz de manipular todos esses problemas. O tratamento de variáveis de valores contínuos é especialmente importante porque tanto processos físicos quanto financeiros fornecem dados numéricos. Vários pacotes comerciais foram construídos para atender a esses critérios, e eles têm sido usados para desenvolver milhares de sistemas de campo. Em muitas áreas da indústria e do comércio, as árvores de decisão costumam ser o primeiro método experimentado quando um método de classificação tem de ser extraído de um conjunto de dados. Uma propriedade importante das árvores de decisão é que é possível para um ser humano entender a razão da saída do algoritmo de aprendizagem (na realidade, essa é uma *exigência* legal para decisões financeiras que estão sujeitas a leis contra a discriminação). Essa propriedade não é compartilhada por algumas outras representações, tais como as redes neurais.

18.4 AVALIAÇÃO E ESCOLHA DA MELHOR HIPÓTESE

Queremos aprender uma hipótese que melhor se ajuste aos dados futuros. Para tornar isso preciso precisamos definir "dados futuros" e "melhor". Façamos a **suposição de estacionaridade**: que há uma distribuição de probabilidade sobre exemplos que permanece estacionária ao longo do tempo. Cada exemplo de ponto de dados (antes de vê-lo) é uma variável aleatória E_j cujo valor observado $e_j = (x_j, y_j)$ é amostrado da distribuição e é independente dos exemplos anteriores:

$$\mathbf{P}(E_j|E_{j-1}, E_{j-2}, \ldots) = \mathbf{P}(E_j) ,$$

e cada exemplo tem uma distribuição de probabilidades anterior idêntica:

$$\mathbf{P}(E_j) = \mathbf{P}(E_{j-1}) = \mathbf{P}(E_{j-2}) = \cdots .$$

Os exemplos que satisfazem essas suposições são chamados *independentes e identicamente distribuídos* ou **i.i.d.** Uma suposição i.i.d. liga o passado ao futuro; sem tal conexão, todas as apostas estão fora — o futuro poderá ser qualquer coisa (veremos mais adiante que a aprendizagem ainda pode ocorrer se houver mudanças lentas na distribuição).

O próximo passo é definir o "melhor ajuste". Definimos a **taxa de erro** de uma hipótese como a proporção de erros que ela comete — a proporção de vezes em que $h(x) \neq y$ para o exemplo (x, y). Agora, só porque uma hipótese h tem taxa de erro baixa no conjunto de treinamento não significa que ela generalize bem. Um professor sabe que um exame não vai avaliar com precisão os alunos se eles já viram as questões do exame. Da mesma forma, para obter avaliação precisa de uma hipótese, é preciso testá-la em um conjunto de exemplos que não tenham sido vistos ainda. A abordagem mais simples é a que já vimos: dividir aleatoriamente os dados disponíveis em um conjunto de treinamento a partir do qual o algoritmo de aprendizagem produz h e um conjunto de teste em que a precisão de h é avaliada. Esse método, chamado às vezes de **validação cruzada por retenção**, tem a desvantagem de não conseguir usar todos os dados disponíveis; se utilizarmos a metade dos dados para o conjunto de teste, estaremos treinando em apenas metade dos dados, e podemos ter uma hipótese mais fraca. Por outro lado, se reservamos apenas 10% dos dados para o conjunto de teste, podemos, por acaso estatístico, obter uma estimativa fraca da precisão real.

Podemos apertar mais os dados e ainda obter uma estimativa precisa usando uma técnica chamada **validação cruzada com *k*-repetições**. A ideia é que cada exemplo sirva duplamente — como dados de treinamento e dados de teste. Primeiro dividimos os dados em k subconjuntos iguais. Em seguida, realizamos k rodadas de aprendizagem; em cada rodada $1/k$ dos dados é retido como um conjunto de teste e os exemplos restantes são usados como dados de treinamento. A pontuação média do conjunto de teste de k rodadas deve então ser uma estimativa melhor do que uma pontuação única. Os valores populares de k são 5 e 10 — o suficiente para dar uma estimativa que é estatisticamente provável que seja precisa, a um custo 5-10 vezes maior do tempo de computação. O extremo é $k = n$, também conhecido como **validação cruzada com omissão de um** ou **VCCOU**.

Apesar dos melhores esforços dos metodólogos estatísticos, os usuários frequentemente invalidam seus resultados ao **espreitar** inadvertidamente os dados de teste. A espreita pode acontecer assim: um algoritmo de aprendizagem tem vários "botões" que podem ser fraudados para ajustar seu comportamento — por exemplo, vários critérios diferentes para escolher o próximo atributo de aprendizagem em árvore de decisão. O pesquisador gera hipóteses para várias configurações

diferentes dos botões, as medidas de suas taxas de erro sobre o conjunto de teste e relatórios de taxa de erro das melhores hipóteses. Infelizmente, ocorreu a espreita! A razão é que a hipótese foi selecionada com base em sua *taxa de erro de conjunto de teste*; assim, a informação sobre o conjunto de teste vazou no algoritmo de aprendizagem.

Espreitar é uma consequência de uso do desempenho do conjunto de teste, tanto para *escolher* uma hipótese como para *avaliá*-la. A maneira de evitar isso é *realmente* reter o conjunto de teste — bloqueá-lo até que a aprendizagem esteja completa e se deseja simplesmente obter uma avaliação independente da hipótese final. (E, então, se você não gostar dos resultados… terá de obter, e bloquear, um conjunto de teste completamente novo se quiser voltar e encontrar uma hipótese melhor.) Se o conjunto de teste estiver bloqueado, mas você ainda quiser medir o desempenho dos dados não vistos, como forma de selecionar uma boa hipótese, divida os dados disponíveis (sem o conjunto de teste) em um conjunto de treinamento e em um **conjunto de validação**. A próxima seção mostra como usar conjuntos de validação para encontrar uma boa contrapartida entre a complexidade da hipótese e a excelência do ajuste.

18.4.1 Seleção do modelo: complexidade *versus* excelência de ajuste

Na Figura 18.1 mostramos que os polinômios de maior grau podem se ajustar melhor aos dados de treinamento, mas quando o grau é muito alto eles vão se superadaptar e desempenhar mal a validação dos dados. A escolha do grau do polinômio é um exemplo do problema de **seleção de modelos**. Imagine a tarefa de encontrar a melhor hipótese como duas tarefas: a seleção do modelo define o espaço de hipóteses e a **otimização** encontra a melhor hipótese dentro desse espaço.

Nesta seção, vamos explicar como escolher entre os modelos que são parametrizados por *tamanho*. Por exemplo, com polinômios temos o *tamanho* = 1 para funções lineares, o *tamanho* = 2 para equações quadráticas, e assim por diante. Para árvores de decisão, o tamanho pode ser o número de nós na árvore. Em todos os casos queremos encontrar o valor do parâmetro *tamanho* que melhor equilibre a subadaptação e a superadaptação para proporcionar a melhor precisão do conjunto de teste.

A Figura 18.8 mostra um algoritmo para realizar a seleção do modelo e a otimização. É um **empacotador** que toma um algoritmo de aprendizagem como argumento (APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO, por exemplo). O empacotador enumera os modelos de acordo com o parâmetro *tamanho*. Para cada tamanho, ele usa validação cruzada em *Aprendiz* para calcular a taxa média de erro sobre os conjuntos de treinamento e de teste. Começaremos com os modelos menores e mais simples (que provavelmente subadaptam os dados) e fazem iteração, considerando os modelos mais complexos em cada etapa, até que os modelos comecem a superadaptar. Na Figura 18.9 observamos curvas típicas: o erro do conjunto de treinamento diminui monotonicamente (embora, em geral, possa haver uma ligeira variação aleatória), enquanto, a princípio, o conjunto de validação de erro diminui e depois aumenta quando o modelo começa a superadaptar. O procedimento de validação cruzada pega o valor de *tamanho* com o menor erro do conjunto de validação, a parte inferior da curva em forma de U. Geramos então uma hipótese desse *tamanho* usando todos os dados (sem reter qualquer deles). Finalmente, é certo que devemos avaliar a hipótese retornada em um

conjunto de teste em separado.

função VALIDAÇÃO-CRUZADA-EMPACOTADOR (*Aprendiz, k, exemplos*) **retorna** *hipótese*
variáveis locais: *errT*, uma tabela, indexada por *tamanho*, armazenamento da taxa de erro do cor
errV, uma tabela, indexada por *tamanho*, armazenamento da taxa de erro do
para tamanho = 1 **até faça**
errT[*tamanho*], *errV*[*tamanho*] ← VALIDAÇÃO-CRUZADA(*Aprendiz, tamanho, k, exemplos*)
se *errT* convergiu **então faça**
melhor-tamanho ← o valor do *tamanho* com *errV*[*tamanho*] mínimo
retornar *Aprendi* (melhor_tamanho*, exemplos*)

função VALIDAÇÃO-CRUZADA(*Aprendiz, tamanho, k, exemplos*) **retornar** dois valores:
taxa de erro média do conjunto-de-treinamento, taxa de erro média do conjunto-de-validação
err_dobraT ← 0; *err_dobraV* ← 0
para dobra = 1 até *k* **faça**
conjunto-de-treinamento, *conjunto-de-validação* ← PARTIÇÃO (*exemplos, dobra, k*)
h ← *Aprendiz* (*tamanho, conjunto-de-treinamento*)
err_dobraT ← *err_dobraT* + TAXA-ERRO(*h*, *conjunto-de-treinamento*)
err_dobraV ← *err_dobraV* + TAXA-ERRO (*h, conjunto-de-validaçao*)
retornar *err_dobra T/k, err_dobraV/k*

Figura 18.8 Algoritmo para selecionar o modelo que tem a menor taxa de erro em validação de dados através da construção de modelos de complexidade crescente e da escolha daquele com melhor taxa empírica de erro na validação de dados. Aqui *errT* significa taxa de erro dos dados de treinamento, e *errV* significa taxa de erro de validação de dados. *Aprendiz*(*tamanho, exemplos*) retorna uma hipótese cuja complexidade é definida pelo *tamanho* do parâmetro e que é instruída pelos exemplos. PARTIÇÃO(*exemplos, dobra, k*) divide *exemplos* em dois subconjuntos: um conjunto de validação de tamanho *N/k* e um conjunto de treinamento com todos os outros exemplos. A divisão é diferente para cada valor de *dobra*.

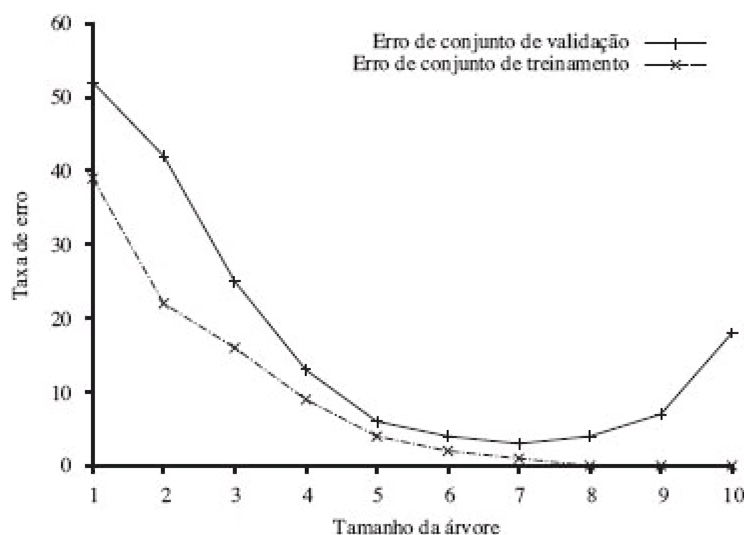


Figura 18.9 Taxas de erro em dados de treinamento (linha inferior, tracejada) e de validação de dados (linha superior, sólida) para árvores de decisão de tamanhos diferentes. Paramos quando a

taxa de erro do conjunto de treinamento tende a ficar assíntota e, então, escolhemos a árvore com erro mínimo no conjunto de validação, nesse caso a árvore com tamanho de sete nós.

Essa abordagem requer que o algoritmo de aprendizagem aceite um parâmetro, *tamanho*, e entregue uma hipótese desse tamanho. Como dissemos, para a aprendizagem em árvore de decisão, o tamanho pode ser o número de nós. Podemos modificar o APRENDIZ- EM-ÁRVORE-DE-DECISÃO para que ele pegue o número de nós como entrada, construa a árvore de busca em largura em vez de em profundidade (mas em cada nível ainda escolha o atributo de maior ganho em primeiro lugar) e pare quando atingir o número desejado de nós.

18.4.2 De taxas de erro a perdas

Até agora, tentamos minimizar a taxa de erro. É claro que isso é melhor do que maximizar a taxa de erro, mas não é toda a história. Considere o problema de classificação de mensagens de e-mail como spam ou não spam. É pior classificar não spam como spam (e perder, portanto, uma mensagem muito importante), do que classificar spam como não spam (e, portanto, sofrer alguns segundos de aborrecimento). Assim, um classificador com taxa de erro de 1%, em que quase todos os erros foram classificar spam como não spam, seria melhor do que um classificador com apenas uma taxa de erro de 0,5%, se a maioria desses erros fosse classificar não spam como spam. Vimos no Capítulo 16 que os tomadores de decisão devem maximizar a utilidade esperada, algo que os aprendizes devem maximizar também. Em aprendizagem de máquina é tradicional expressar utilidade por meio da **função de perda**. A função de perda $L(x, y, \hat{y})$ é definida como o montante de utilidade perdida pela previsão $h(x) = \hat{y}$ quando a resposta correta é $f(x) = y$:

$$\begin{aligned} L(x, y, \hat{y}) = & \ Utilidade \text{ (resultado do uso de } y \text{ dada uma entrada } x) \\ & - Utilidade \text{ (resultado do uso de } \hat{y} \text{ dada uma entrada } x) \end{aligned}$$

Essa é a formulação mais geral da função de perda. Muitas vezes utiliza-se uma versão simplificada, $L(y, \hat{y})$, que é independente de x. Usaremos a versão simplificada no restante do capítulo, o que significa que não podemos dizer que seja pior classificar erroneamente a carta da mãe do que a carta de um primo chato, mas podemos dizer que é 10 vezes pior classificar não spam como spam do que vice-versa:

$$L(spam, \text{ } não \text{ } spam) = 1, \qquad L(não \text{ } spam, \text{ } spam) = 10.$$

Observe que $L(y, y)$ é sempre zero; por definição, não há perda quando você supõe exatamente o correto. Para funções com saídas discretas, podemos enumerar um valor de perda para cada erro de classificação possível, mas não podemos enumerar todas as possibilidades para dados com valores reais. Se $f(x)$ fosse 137.035999, ficaríamos bastante satisfeitos com $h(x)$ = 137.036, mas o quanto felizes deveríamos estar? Em geral, pequenos erros são melhores do que grandes; duas funções que implementam essa ideia são o valor absoluto da diferença (chamado de perda L_1) e o quadrado da diferença (chamado de perda L_2). Se estivermos satisfeitos com a ideia de minimizar a taxa de erro, poderemos usar a função perda $L_{0/1}$, que tem perda de 1 para uma resposta incorreta e é apropriada

para saídas de valores discretos:

Perda de valor absoluto: $L_1(y, \hat{y}) = |y - \hat{y}|$
Perda de erro quadrático: $L_2(y, \hat{y}) = (y - \hat{y})^2$
Perda de 0/1: $L_{0/1}(y, \hat{y}) = 0$ se $y = \hat{y}$, senão 1

O agente de aprendizagem pode, teoricamente, maximizar a sua utilidade esperada escolhendo a hipótese que minimiza a perda esperada de todos os pares de entrada e saída que vai verificar. Não tem sentido falar sobre essa expectativa sem definir uma distribuição de probabilidade anterior, **P**(*X*, *Y*) para os exemplos. Seja ε o conjunto de todos os possíveis exemplos de entrada e saída. Então, a **generalização da perda** esperada de uma hipótese *h* (com relação à função de perda *L*) é

$$GenPerda_L(h) = \sum_{(x,y)\in\varepsilon} L(y, h(x))\, P(x,y),$$

e a melhor hipótese, h^*, é a que apresenta a mínima perda de generalização esperada:

$$h^* = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\; GenPerda_L(h).$$

Como *P(x, y*) não é conhecido, o agente de aprendizagem só pode estimar a perda de generalização com **perda empírica** sobre um conjunto de exemplos, *E*:

$$PerdaEmp_{L,E}(h) = \frac{1}{N} \sum_{(x,y)\in E} L(y, h(x)).$$

A melhor hipótese estimada $\hat{h}^*$ é, então, a que tem a perda empírica mínima:

$$\hat{h}^* = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\; PerdaEmp_{L,E}(h).$$

Existem quatro razões pelas quais $\hat{h}^*$ pode ser diferente da verdadeira função *f*: impossibilidade de realizar, variância, ruído e complexidade computacional. Primeiro, *f* pode não ser realizável — pode não estar em $\mathcal{H}$ — ou pode estar presente de tal forma que outras hipóteses sejam preferidas. Segundo, um algoritmo de aprendizagem retornará hipóteses diferentes para conjuntos de exemplos diferentes, mesmo que esses conjuntos sejam extraídos da mesma função *f* verdadeira, e essas hipóteses vão fazer previsões diferentes em novos exemplos. Quanto maior a variância entre as previsões, maior a probabilidade de erro significativo. Note que, mesmo quando o problema é realizável, ainda haverá variância aleatória, mas essa variância tende a zero à medida que o número de exemplos de treinamento aumenta. Terceiro, *f* pode ser não determinístico ou **ruidoso** — pode retornar valores diferentes para *f(x*) *a* cada vez que *x* ocorre. Por definição, o ruído não pode ser previsto; em muitos casos, surge porque os rótulos *y* observados são o resultado de atributos do ambiente que não constam em *x*. E, finalmente, quando $\mathcal{H}$ é complexo, pode ser intratável computacionalmente para pesquisar de forma sistemática todo o espaço de hipótese. O melhor que podemos fazer é uma busca local (subidas de encosta ou busca gulosa) que explora apenas uma parte do espaço. Isso nos dá um erro de aproximação. Combinando as fontes de erro, ficamos com uma estimativa de aproximação da função verdadeira *f*.

Os métodos tradicionais em estatística e os primeiros anos de aprendizagem de máquina concentraram-se em **aprendizagem em pequena escala**, em que o número de exemplos de treinamento variava de dezenas a poucos milhares. Aqui o erro de generalização em sua maioria vinha do erro de aproximação de não ter o *f* verdadeiro no espaço de hipóteses e do erro de estimativa de não ter exemplos suficientes de treinamento para limitar a variância. Nos últimos anos tem havido mais ênfase em **aprendizagem em larga escala**, muitas vezes com milhões de exemplos. Aqui o erro de generalização é dominado pelos limites do cálculo: há dados suficientes e um modelo abundante o suficiente para podermos encontrar um *h* muito próximo do *f* verdadeiro, mas o cálculo para chegar a isso é demasiado complexo; por isso, estabelecemos uma aproximação subótima.

18.4.3 Regularização

Na Seção 18.4.1, vimos como fazer seleção de modelo com validação cruzada sobre o tamanho do modelo. Uma abordagem alternativa é a busca de uma hipótese que minimize diretamente a soma ponderada da perda empírica e a complexidade da hipótese, que chamaremos de custo total:

$$\begin{aligned} Custo(h) &= PerdaEmp(h) + \lambda\ Complexidade(h) \\ \hat{h}^* &= Custo(h) \underset{h \in \mathcal{H}}{\text{argmin}}. \end{aligned}$$

Aqui λ é um parâmetro, um número positivo que serve como taxa de conversão entre perda e complexidade de hipótese (que, afinal, não são medidas na mesma escala). Essa abordagem combina perda e complexidade em uma métrica, permitindo-nos encontrar a melhor hipótese. Infelizmente ainda temos de fazer busca de validação cruzada para encontrar a hipótese que generaliza melhor, mas dessa vez é com valores diferentes de λ em vez de *tamanho*. Selecionamos o valor de λ que nos dá a melhor pontuação do conjunto de validação.

O processo de penalizar explicitamente a hipótese complexa é chamado de **regularização** (porque procura por uma função que seja mais regular ou menos complexa). Observe que a função de custo nos obriga a fazer duas escolhas: a função perda e a medida de complexidade, que é chamada de função de regularização. A escolha da função de regularização depende do espaço de hipótese. Por exemplo, uma função de regularização para polinômios é a soma dos quadrados dos coeficientes — a manutenção da soma pequena nos levaria para longe dos polinômios sinuosos da Figura 18.1(b) e (c). Vamos mostrar um exemplo desse tipo de regularização na Seção 18.6.

Outra maneira de simplificar os modelos é reduzir as dimensões com as quais os modelos funcionam. Um processo de **seleção de atributos** pode ser realizado para descartar atributos que parecem ser irrelevantes. A poda χ^2 é uma espécie de seleção de atributos.

De fato, é possível obter perda empírica e a complexidade medida na mesma escala sem o fator de conversão λ: ambos podem ser medidos em bits. Primeiro codifique a hipótese como um programa de máquina de Turing e conte o número de bits. Em seguida, conte o número de bits necessários para codificar os dados, em que um exemplo previsto corretamente custa zero bits e o custo de um exemplo previsto incorretamente depende da grandeza do erro. O **comprimento de descrição mínimo** ou hipótese CDM minimiza o número total de bits necessário. Isso funciona bem no limite, mas para problemas menores há uma dificuldade pois a escolha da codificação para o programa —

por exemplo, qual a melhor forma de codificar uma árvore de decisão como sequência de bits — afeta o resultado. No Capítulo 20, descrevemos uma interpretação probabilística da abordagem CDM.

18.5 TEORIA DA APRENDIZAGEM

A principal questão sem resposta na aprendizagem é esta: como podemos ter certeza de que o algoritmo de aprendizagem produziu uma hipótese que vai prever o valor correto para as entradas não vistas anteriormente? Em termos formais, como sabemos que a hipótese h está próxima da função-alvo f se não sabemos o que é f? Essas questões têm sido ponderadas por vários séculos. Em décadas mais recentes, outras questões surgiram: quantos exemplos precisamos para obter um bom h? Que espaço de hipótese devemos usar? Se o espaço de hipótese for muito complexo, podemos ainda achar o melhor h ou temos que nos contentar com um máximo local no espaço de hipóteses? Quanto complexo deve ser h? Como podemos evitar a superadaptação? Esta seção examina essas questões.

Vamos começar com a pergunta de quantos exemplos são necessários para a aprendizagem. Vimos, a partir da curva de aprendizagem em árvore de decisão sobre o problema do restaurante (Figura 18.7), que melhora com mais dados de treinamento. As curvas de aprendizagem são úteis, mas específicas para determinado algoritmo de aprendizagem em um problema particular. Existem alguns princípios mais genéricos que regem o número de exemplos que em geral são necessários? Perguntas como essa são abordadas pela **teoria da aprendizagem computacional**, que fica no cruzamento entre IA, estatística e ciência da computação teórica. O princípio subjacente é que *qualquer hipótese que esteja seriamente errada será quase certamente "descoberta" com alta probabilidade depois de um pequeno número de exemplos, porque vai fazer uma previsão incorreta. Assim, qualquer hipótese que seja consistente com um conjunto suficientemente grande de exemplos de conjunto de treinamento é improvável de estar seriamente errada, ou seja, deve estar* ***provavelmente aproximadamente correta.*** Qualquer algoritmo de aprendizagem que retorne hipóteses que sejam provavelmente aproximadamente correto é chamado de algoritmo de **aprendizagem PAC**; podemos usar essa abordagem para fornecer limites sobre o desempenho de vários algoritmos de aprendizagem.

Os teoremas de aprendizagem PAC, como os demais teoremas, são consequências lógicas dos axiomas. Quando um *teorema* (em oposição, digamos, a um comentarista político) afirma algo sobre o futuro com base no passado, os axiomas têm de fornecer a "energia" para fazer essa ligação. Para a aprendizagem PAC, a ligação é fornecida pelo pressuposto de estacionariedade já introduzido, que diz que exemplos futuros vão ser retirados da mesma distribuição fixa $\mathbf{P}(E) = \mathbf{P}(X, Y)$ como os exemplos passados (observe que não temos de saber que distribuição é, apenas que ela não muda). Além disso, para manter as coisas simples, vamos supor que a função verdadeira f seja determinística e membro da classe de hipótese $\mathcal{H}$ que está sendo considerada.

Os teoremas PAC mais simples lidam com funções booleanas, para as quais a perda 0/1 é apropriada. A **taxa de erro** de uma hipótese h, definida antes informalmente; aqui é definida formalmente como o erro de generalização esperado para exemplos tirados da distribuição

estacionária:

$$\text{error}(h) = GenPerda_{0/1}(h) = \sum_{x,y} L_{0/1}(y, h(x))\, P(x,y).$$

Em outras palavras, o erro (h) é a probabilidade de que h classifique incorretamente um novo exemplo. Essa é a mesma quantidade que estava sendo medida experimentalmente pelas curvas de aprendizagem mostradas anteriormente.

A hipótese h é chamada de **aproximadamente correta** se o erro $(h) \leq \in$ onde $\in$ é uma constante pequena. Vamos mostrar que podemos encontrar N tal que, depois de verificar N exemplos, com probabilidade alta, todas as hipóteses consistentes serão aproximadamente corretas. Pode-se pensar em uma hipótese aproximadamente correta como sendo "próxima" à função verdadeira no espaço de hipótese: repousa sobre o que é chamado **$\in$-bola** em torno da verdadeira função f. O espaço de hipótese fora dessa esfera é chamado $\mathcal{H}_{\text{ruim}}$.

Podemos calcular a probabilidade de que uma hipótese "seriamente errada" $h_r \in \mathcal{H}_{\text{ruim}}$ seja consistente com os primeiros N exemplos, como segue. Sabemos que o erro $(h_r) > \in$. Assim, a probabilidade de que esteja de acordo com um exemplo dado é no máximo $1 - \in$. Uma vez que os exemplos são independentes, o limite para N exemplos é

$$P(h_r \text{ concorda com } N \text{ exemplos}) \leq (1 - \in)^N.$$

A probabilidade de que $\mathcal{H}_{\text{ruim}}$ contenha pelo menos uma hipótese consistente é limitada pela soma das probabilidades individuais:

P($\mathcal{H}_{\text{ruim}}$ contém uma hipótese consistente) $\leq |\mathcal{H}_{\text{ruim}}|(1 - \in)^N \leq |\mathcal{H}|(1 - \in)^N$, onde usamos o fato de que $|\mathcal{H}_{\text{ruim}}| \leq |\mathcal{H}|$. Gostaríamos de reduzir a probabilidade desse evento abaixo de algum número pequeno δ:

$$|\mathcal{H}|(1 - \epsilon)^N \leq \delta\,.$$

Dado que $1 - \epsilon \leq e^{-\epsilon}$, podemos conseguir isso se permitirmos que o algoritmo verifique

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|\mathcal{H}|\right) \tag{18.1}$$

exemplos. Assim, se um algoritmo de aprendizagem retornar uma hipótese que seja consistente com essa quantidade de exemplos, com probabilidade de pelo menos $1 - \delta$ existe erro de, no máximo, $\in$. Em outras palavras, provavelmente é aproximadamente correto. O número de exemplos necessários, em função de $\in$ e δ, é chamado de **complexidade da amostra** do espaço de hipótese.

Como vimos anteriormente, se $\mathcal{H}$ for o conjunto de todas as funções booleanas sobre n atributos, então $|\mathcal{H}| = 2^{2^n}$. Assim, a complexidade da amostra do espaço cresce 2^n. Como o número de exemplos possíveis também é 2^n, isso sugere que a aprendizagem PAC na classe de todas as funções booleanas requer verificação de todos ou quase todos os exemplos possíveis. Pensando rapidamente sabemos a razão para isso: $\mathcal{H}$ contém hipóteses suficientes para classificar qualquer conjunto de

exemplos dados de todas as maneiras possíveis. Em particular, para qualquer conjunto de N exemplos, o conjunto de hipóteses consistentes com esses exemplos contém números iguais de hipóteses que predizem que x_{N+1} seja positivo e de hipóteses que predizem que x_{N+1} seja negativo.

Então, para obter a generalização real de exemplos não vistos, parece que precisamos restringir o espaço de hipótese $\mathcal{H}$ de alguma forma, mas é claro que, se restringirmos o espaço, também poderemos eliminar a função verdadeira. Há três maneiras de fugir desse dilema. A primeira, a qual será abordada no Capítulo 19, é trazer o conhecimento prévio para apoiar o problema. A segunda, que introduzimos na Seção 18.4.3, é insistir que o algoritmo retorne não apenas qualquer hipótese consistente, mas de preferência uma simples (como é feito em aprendizagem em árvore de decisão). Nos casos em que é tratável encontrar hipóteses consistentes simples, os resultados da complexidade da amostra são geralmente melhores do que os da análise com base apenas na consistência. A terceira maneira, que veremos a seguir, é concentrar-se em subconjuntos aprendíveis do espaço de hipótese inteiro de funções booleanas. Essa abordagem se fundamenta no pressuposto de que a linguagem restrita contém uma hipótese h que está perto o suficiente da função verdadeira f; os benefícios são que o espaço de hipótese restrito permite a generalização eficaz e geralmente é mais fácil de buscar. Examinaremos agora com mais detalhes essa linguagem restrita.

18.5.1 Exemplo de aprendizagem PAC: aprendizagem de listas de decisão

Vamos agora mostrar como aplicar a aprendizagem PAC para um espaço de hipótese novo: **listas de decisão**. Uma lista de decisão consiste em uma série de testes, cada um dos quais é uma conjunção de literais. Se um teste for bem-sucedido quando aplicado a uma descrição do exemplo, a lista de decisão especificará o valor a ser retornado. Se o teste falhar, o processamento continua com o próximo teste na lista. As listas de decisão se assemelham às árvores de decisão, mas a sua estrutura geral é mais simples: elas se ramificam apenas em uma direção. Em contraste, os testes individuais são mais complexos. A Figura 18.10 mostra uma lista de decisão que representa a hipótese a seguir:

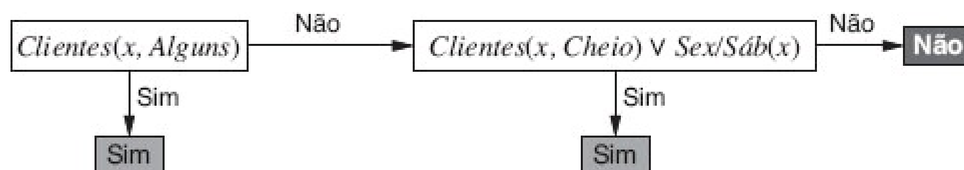


Figura 18.10 Lista de decisão para o problema do restaurante.

$$VaiEsperar \Leftrightarrow (Clientes = Alguns) \lor (Clientess = Cheio \land Sex/Sáb).$$

Se permitirmos testes de tamanho arbitrário, as listas de decisão podem representar qualquer função booleana (Exercício 18.14). Por outro lado, se restringirmos o tamanho de cada teste para no máximo k literais, é possível para o algoritmo de aprendizagem generalizar com sucesso um pequeno número de exemplos. Chamamos essa linguagem de k-**DL**. O exemplo da Figura 18.10 está em 2-DL. É fácil mostrar (Exercício 18.14) que k-DL inclui como subconjunto a linguagem k-**DT**, o conjunto de todas as árvores de decisão de profundidade, no máximo, k. É importante lembrar que a linguagem

particular a que se refere o k-DL depende dos atributos utilizados para descrever os exemplos. Usaremos a notação k-DL(n) para indicar uma linguagem k-DL usando n atributos booleanos.

A primeira tarefa é mostrar que k-DL é aprendível, ou seja, que qualquer função em k-DL pode ser aproximada com precisão após o treinamento com um número razoável de exemplos. Para fazer isso, precisamos calcular o número de hipóteses na linguagem. Seja a linguagem de testes — conjunções de, no máximo, k literais usando n atributos — $Conj(n, k)$. Como uma lista de decisão é constituída por testes, e cada teste pode estar ligado a um resultado *Sim ou Não* ou pode estar ausente da lista de decisão, há no máximo $3^{|Conj\ (n,k)|}$ conjuntos distintos de testes de componente. Cada um desses conjuntos de testes pode estar em qualquer ordem, de modo que

$$|k\text{-DL}(n)| \leq 3^{|Conj(n,k)|}|Conj(n,k)|!\,.$$

O número de conjunções de k literais de n atributos é dado por

$$|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = O(n^k)\,.$$

Assim, após algum trabalho, obtemos

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}\,.$$

Podemos ligá-la à Equação 18.1 para mostrar que o número de exemplos necessários para a aprendizagem PAC em uma função k-DL é polinomial em n:

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + O(n^k \log_2(n^k))\right)\,.$$

Portanto, qualquer algoritmo que retorne uma lista de decisão consistente vai PAC-aprender uma função k-DL em um número razoável de exemplos, para pequenos k.

A próxima tarefa é encontrar um algoritmo eficiente que retorne uma lista de decisão consistente. Usaremos um algoritmo guloso chamado APRENDIZAGEM-EM-LISTA-DE-DECISÃO que repetidamente encontra um teste que concorda exatamente com um subconjunto do conjunto de treinamento. Uma vez que ele encontra um teste desse tipo, adiciona-o à lista de decisão em construção e remove os exemplos correspondentes. Constrói, então, o restante da lista de decisão utilizando apenas os exemplos restantes. Repete-se até que não haja mais exemplos. O algoritmo é mostrado na Figura 18.11.

função APRENDIZAGEM-LISTA-DE-DECISÃO (*exemplos*) **retorna** uma lista de decisão ou *falha*
  se *exemplos* é vazio **então retornar** lista de decisão comum *Não*
  t ← um teste que combina um subconjunto não vazio de $exemplos_t$ de *exemplos*
    tal que os membros de $exemplos_t$ são todos positivos ou negativos
  se não houver tal t **então retornar** *falha*

se exemplos em $exemplos_t$ é positivo **então** $o \leftarrow Sim$ **senão** $o \leftarrow Não$
retornar uma lista de decisão com o teste t inicial e o resultado o e os testes restantes dados por
APRENDIZAGEM- EM- LISTA-DE-DECISÃO($exemplos$ — $exemplos_t$)

Figura 18.11 Algoritmo de aprendizagem em lista de decisão.

Esse algoritmo não especifica o método para selecionar o próximo teste para adicionar à lista de decisão. Embora os resultados formais dados anteriormente não dependam do método de seleção, parece razoável preferir pequenos testes que correspondam a grandes conjuntos de exemplos classificados de forma uniforme, de modo que a lista de decisão global seja a mais compacta possível. A estratégia mais simples é encontrar o menor teste t que combine com qualquer subconjunto uniformemente classificado, independentemente do tamanho do subconjunto. Essa abordagem também funciona muito bem, como a Figura 18.12 sugere.

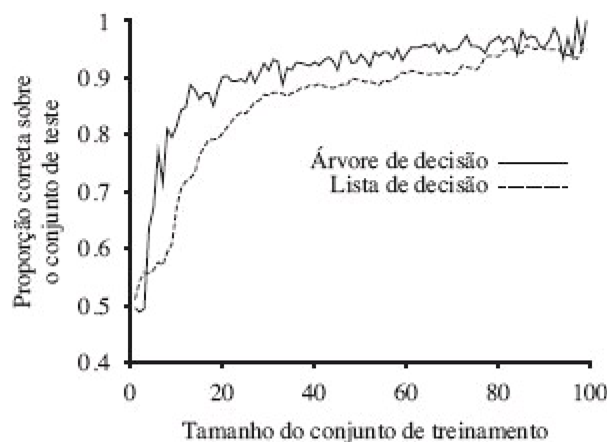


Figura 18.12 Curva de aprendizagem para o algoritmo APRENDIZAGEM-EM -LISTA-DE-DECISÃO sobre os dados do restaurante. A curva de aprendizagem para a APRENDIZAGEM-EM-ÁRVODE-DE-DECISÃO é mostrada para comparação.

18.6 REGRESSÃO E CLASSIFICAÇÃO COM MODELOS LINEARES

Agora é hora de passar de árvores de decisão e de listas para um espaço de hipótese diferente, que tem sido utilizado durante centenas de anos: a classe de **funções lineares** de entradas de valor contínuo.

Vamos começar com o caso mais simples: regressão com uma função linear univariada, também conhecida como "adaptação de uma linha reta". A Seção 18.6.2 abrange o caso multivariado. As Seções 18.6.3 e 18.6.4 mostram como transformar funções lineares em classificadores aplicando limiares duros e suaves.

18.6.1 Regressão linear univariada

Uma função linear univariada (linha reta), com entrada x e saída y tem a forma $y = w_1 x + w_0$, onde w_0 e w_1 são coeficientes de valores reais a serem aprendidos. Usamos a letra w porque imaginamos os coeficientes como **pesos;** o valor de y é alterado pela mudança do peso relativo de um termo para outro. Definimos $\mathbf{w}$ como o vetor $[w_0, w_1]$ e definimos

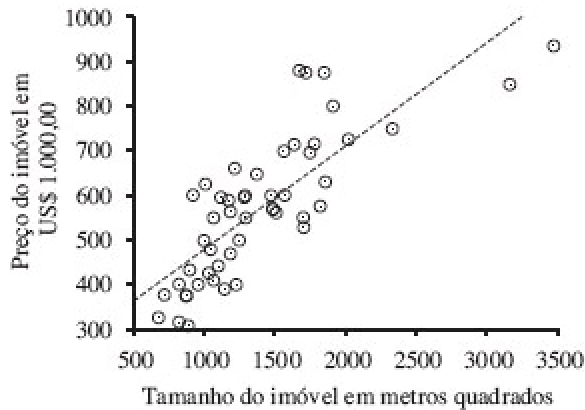
$$h_{\mathbf{w}}(x) = w_1 x + w_0.$$

A Figura 18.13(a) mostra um exemplo de conjunto de treinamento de n pontos no plano x, y, cada ponto representando o tamanho em metros quadrados e o preço de uma casa à venda. A tarefa de encontrar o $h_{\mathbf{w}}$ que melhor se encaixe nesses dados é chamada de **regressão linear**. Para ajustar uma linha com os dados, tudo o que temos a fazer é encontrar os valores dos pesos $[w_0, w_1]$ que minimizam a perda empírica. É tradicional (voltando a Gauss[3]) usar a função de perda quadrática, L_2, a soma de todos os exemplos de treinamento:

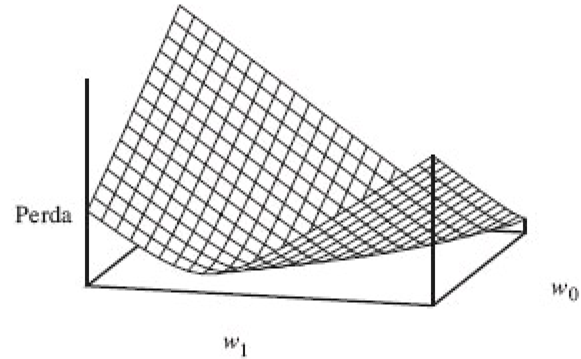
$$Perda(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 .$$

Gostaríamos de encontrar $\mathbf{w}^* = \text{argmin}_{\mathbf{w}}\, Perda(h_{\mathbf{w}})$. A soma $\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$ é minimizada quando suas derivadas parciais em relação a w_0 e w_1 são zero:

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0 \quad \text{e} \quad \frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0 . \qquad (18.2)$$



(a)



(b)

Figura 18.13 (a) Pontos de dados de preço *versus* espaço dos imóveis em área para venda em Berkeley, CA, em julho de 2009, junto com a hipótese de função linear que minimiza o erro de perda ao quadrado: $y = 0{,}232x + 246$. (b) Representação gráfica da função perda $\sum_j (w_1 x_j + w_0 - y_j)^2$ para diversos valores de w_0 e w_1. Observe que a função perda é convexa, com um único mínimo global.

Essas equações têm solução única:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}; \quad w_0 = (\sum y_j - w_1(\sum x_j))/N . \qquad (18.3)$$

Para o exemplo na Figura 18.13(a), a solução é $w_1 = 0{,}232$, $w_0 = 246$, e a linha com os pesos é mostrada na figura como uma linha tracejada.

Muitas formas de aprendizagem envolvem o ajuste de pesos para minimizar a perda, por isso ajuda ter uma imagem mental do que está acontecendo no **espaço de peso** — o espaço definido por todas as configurações de pesos possíveis. Para a regressão linear univariada, o espaço definido por w_0 e w_1 é bidimensional, para que possamos colocar em gráfico a perda em função de w_0 e w_1 em um plano 3-D (veja a Figura 18.13(b)). Observamos que a função perda é **convexa**, conforme definido anteriormente; isso é verdade para *cada* problema de regressão linear com uma função de perda L_2, e implica que não há mínimo local. Em certo sentido, esse é o fim da história para os modelos lineares; se precisarmos ajustar linhas aos dados, aplicamos a Equação 18.3.[4]

Para ir além dos modelos lineares, teremos que encarar o fato de que as equações que definem a perda mínima (como a Equação 18.2), muitas vezes, não têm solução de forma fechada. Em vez disso, se tem de enfrentar um problema de busca de otimização geral em um espaço de peso contínuo. Como indicado na Seção 4.2, tais problemas podem ser resolvidos por um algoritmo de subida de encosta, que segue o **gradiente** da função a ser otimizada. Nesse caso, como estamos tentando minimizar a perda, usaremos a **descida pelo gradiente**. Escolhemos qualquer ponto de partida em espaço de peso — aqui, um ponto no plano (w_0, w_1) — e em seguida movemos para um ponto vizinho que está em declive, repetindo até que convirja para a mínima perda possível:

w ← qualquer ponto no espaço de parâmetros
laço até convergência **faça** (18.4)
para cada *wi* **em w faça**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \, Perda(\mathbf{w})$$

O parâmetro, que chamamos de **tamanho do passo** na Seção 4.2, é geralmente chamado de **taxa de aprendizagem** quando estamos tentando minimizar a perda em um problema de aprendizagem. Pode ser uma constante fixa ou pode decair ao longo do tempo à medida que o processo de aprendizagem prossegue.

Para a regressão univariada, a função de perda é uma função quadrática, então a derivada parcial será uma função linear. (O único cálculo que é necessário saber é $\frac{\partial}{\partial x}x^2 = 2x$ e $\frac{\partial}{\partial x}x = 1$). Primeiro vamos calcular as derivadas parciais — as inclinações — no caso simplificado de apenas um exemplo de treinamento, (x, y):

$$\begin{aligned} \frac{\partial}{\partial w_i} \, Perda\ (\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0))\,, \end{aligned} \qquad (18.5)$$

aplicando para w_0 e w_1 obtemos:

$$\frac{\partial}{\partial w_0} \textit{Perda}\ (\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))\,; \qquad \frac{\partial}{\partial w_1} \textit{Perda}\ (\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x\ .$$

Então, ligando de volta na Equação 18.4 e incorporando 2 em uma taxa de aprendizagem não especificada, temos a seguinte regra de aprendizagem para os pesos:

$$w_0 \leftarrow w_0 + \alpha\,(y - h_{\mathbf{w}}(x))\,; \quad w_1 \leftarrow w_1 + \alpha\,(y - h_{\mathbf{w}}(x)) \times x\,.$$

Essas atualizações fazem sentido intuitivo: se h_{w} (x)> y, ou seja, a saída da hipótese for muito grande, reduza um pouco w_0 e reduza w_1 se x for uma entrada positiva mas aumente w_1 se x for uma entrada negativa.

As equações anteriores abrangem um exemplo de treinamento. Para N exemplos de treinamento, queremos minimizar a soma das perdas individuais para cada exemplo. A derivada de uma soma é a soma das derivadas, por isso temos:

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))\,; \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j\,.$$

Essas atualizações constituem a regra de aprendizagem da **descida pelo gradiente em lotes** para regressão linear univariada. A convergência é garantida para o único mínimo global (contanto que escolhamos um pequeno o suficiente), mas pode ser muito lenta: temos de percorrer todos os dados de treinamento para cada etapa, e pode haver muitas etapas.

Existe outra possibilidade, chamada de **descida estocástica pelo gradiente**, onde consideramos um único ponto de treinamento por vez, seguindo uma etapa após a outra, usando a Equação 18.5. A descida estocástica pelo gradiente pode ser usada em um ambiente on-line, onde chegam novos dados, um de cada vez, ou off-line, onde percorremos os mesmos dados tantas vezes quantas forem necessárias, selecionando uma etapa depois de considerar cada exemplo. Muitas vezes é mais rápido do que a descida pelo gradiente em lotes. Uma taxa de aprendizagem fixa, no entanto, não garante a convergência, podendo oscilar em torno do mínimo, sem se fixar. Em alguns casos, como veremos mais adiante, planejar um cronograma de taxas de aprendizagem descendente (como em recozimento simulado) garante a convergência.

18.6.2 Regressão linear multivariada

Podemos facilmente estender para problemas de regressão linear multivariada, em que cada exemplo $\mathbf{x}_j$ é um vetor de n elementos.[5] Nosso espaço de hipótese é o conjunto de funções da forma

$$h_{sw}(\mathbf{x}_j) = w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}\,.$$

O termo w_0, a interseção, distingue-se como diferente dos outros. Podemos corrigir isso pela criação de um atributo de entrada fictício, $x_{j,0}$, que sempre é definido como igual a 1. Então, h é

simplesmente o produto escalar dos pesos e do vetor de entrada (ou, de forma equivalente, o produto da matriz transposta dos pesos pelo vetor de entrada):

$$h_{sw}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^\top \mathbf{x}_j = \sum_i w_i x_{j,i}\,.$$

O melhor vetor de pesos, **w***, minimiza a perda de erro quadrático nos exemplos:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j)\,.$$

A regressão linear multivariada não é muito mais complicada do que o caso univariado já abrangido. A descida pelo gradiente vai atingir o mínimo (único) da função de perda; a equação de atualização de cada peso w_i é

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i}(y_j - h_{\mathbf{w}}(\mathbf{x}_j))\,. \qquad (18.6)$$

Também é possível resolver analiticamente para o **w** que minimiza a perda. Seja **y** o vetor de saída para os exemplos de treinamento e **X** a **matriz de dados**, ou seja, a matriz de entradas com um exemplo *n* dimensional por linha. Então, a solução

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

minimiza o erro quadrático.

Com a regressão linear univariada não precisamos nos preocupar com a superadaptação. Mas, com a regressão linear multivariada em espaços de dimensão superior, é possível que alguma dimensão que seja realmente irrelevante pareça ser útil por acaso, resultando em **superadaptação**.

Assim, é comum o uso de **regularização** em funções lineares multivariadas para evitar a superadaptação. Lembre-se de que, com a regularização, minimizamos o custo total de uma hipótese, contando tanto com a perda empírica como com a complexidade da hipótese:

Custo(*h*) = *PerdaEmp*(*h*) + λ *Complexidade*(*h*).

Para as funções lineares, a complexidade pode ser especificada em função dos pesos. Podemos considerar uma família de funções de regularização:

$$Complexidade(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q\,.$$

Tal como acontece com as funções de perda[6] com $\theta = 1$ temos a regularização L_1, o que minimiza a soma dos valores absolutos; com $\theta = 2$, a regularização L_2 minimiza a soma dos quadrados. Qual função de regularização se deve escolher? Isso depende do problema específico, mas a regularização L_1 tem uma vantagem importante: tende a produzir um **modelo esparso**. Isto é, muitas vezes define

muitos pesos para zero, declarando efetivamente os atributos correspondentes como irrelevantes — como a APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO faz (embora por um mecanismo diferente). As hipóteses que descartam atributos podem ser mais fáceis de um ser humano entender, e podem ser menos prováveis de superadaptar.

A Figura 18.14 fornece uma explicação intuitiva da razão pela qual a regularização L_1 leva a pesos zero, enquanto a regularização L_2 não leva. Observe que a minimização *Perdas(***w***)* + λ *Complexidade(***w***)* é equivalente à minimização *Perda*(**w**) sujeita à restrição de que *Complexidade*(**w**) $\leq c$, para alguma constante c que esteja relacionada com λ. Agora, na Figura 18.14(a) a caixa em forma de diamante representa o conjunto de pontos **w** no espaço de peso bidimensional que tem a complexidade L_1 menor que c; nossa solução terá de estar em algum lugar dentro dessa caixa. As ovais concêntricas representam contornos da função de perda, com a perda mínima ao centro. Queremos encontrar o ponto na caixa que esteja mais próximo ao mínimo; você pode observar no diagrama que, para uma posição arbitrária de mínimo e seus contornos, será comum para o canto da caixa encontrar sua forma mais próxima ao mínimo só porque os cantos são pontudos. E, claro, os cantos são os pontos que têm valor de zero em alguma dimensão. Na Figura 18.14(b), fizemos o mesmo para a medida de complexidade L_2, que representa um círculo, em vez de um diamante. Aqui você pode verificar que, em geral, não há razão para a interseção aparecer em um dos eixos; assim, a regularização L_2 não tende a produzir pesos zero. O resultado é que o número de exemplos necessários para encontrar um bom h é linear no número de características irrelevantes para a regularização L_2, mas somente com a regularização logarítmica L_1. A evidência empírica de muitos problemas reforça essa análise.

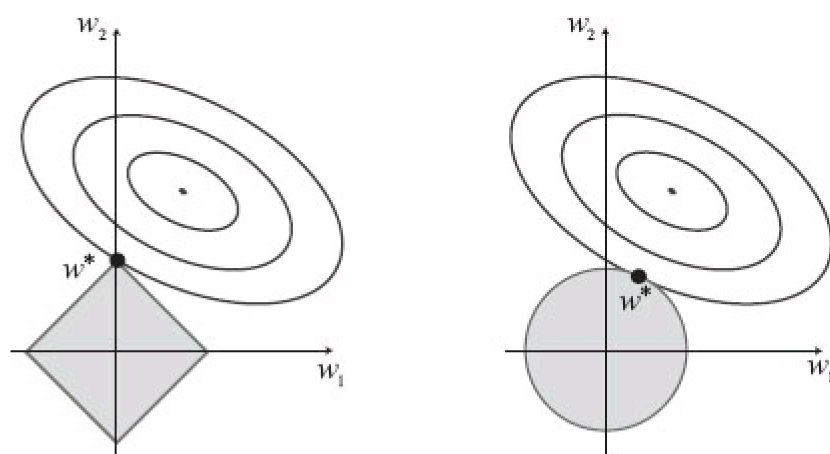


Figura 18.14 Por que a regularização de L_1 tende a produzir um modelo esparso. (a) Com a regularização (caixa) de L_1, a perda mínima atingível (contornos concêntricos), muitas vezes, ocorre em um eixo, o que significa peso zero. (b) Com a regularização L_2 (círculo), é provável que a perda mínima ocorra em qualquer parte do círculo, não dando preferência a pesos zero.

Outra maneira de ver isso é que a regularização L_1 leva os eixos dimensionais a sério, enquanto a L_2 trata-os como arbitrários. A função de L_2 é esférica, o que a torna rotacionalmente invariante: imagine um conjunto de pontos em um plano, medido por suas coordenadas x e y. Agora imagine a rotação de 45º dos eixos. Você gostaria de obter um conjunto de valores diferentes de (x′, y′) representando os mesmos pontos. Se aplicar a regularização L_2 antes e depois da rotação, terá

exatamente o mesmo ponto como resposta (embora o ponto seja descrito com a nova coordenada (x', y')). Isso é apropriado quando a escolha dos eixos realmente for arbitrária — quando não importar se as suas duas dimensões são as distâncias norte e leste ou nordeste e sudeste. Com a regularização L_1 obtém-se uma resposta diferente porque a função L_1 não é rotacionalmente invariante. Isso é apropriado quando os eixos não são intercambiáveis, mas não faz sentido rodar o "número de banheiros" 45º para o "tamanho do lote".

18.6.3 Classificadores lineares com limiar rígido

As funções lineares podem ser usadas tanto para fazer a classificação como a regressão. Por exemplo, a Figura 18.15(a) mostra os pontos de dados de duas classes: os terremotos (que são de interesse para os sismólogos) e as explosões subterrâneas (que são de interesse dos especialistas em controle de armas). Cada ponto é definido por dois valores de entrada, x_1 e x_2, que se referem a magnitudes de onda do corpo e da superfície, calculados a partir do sinal sísmico. Tendo em conta esses dados de treinamento, a tarefa de classificação é aprender uma hipótese h, que terá novos (x_1, x_2) pontos e retornará 0 para terremotos ou 1 para explosões.

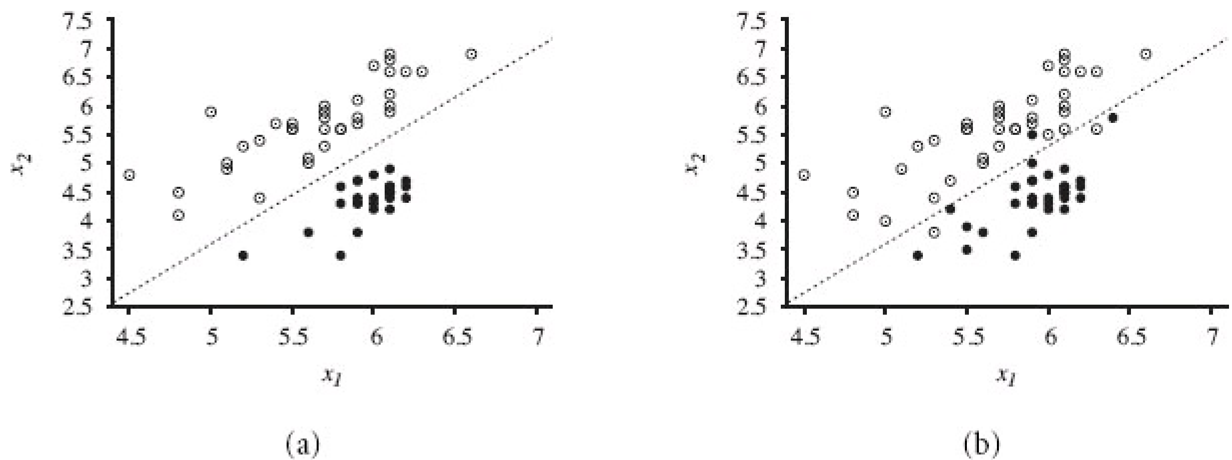


Figura 18.15 (a) Representação gráfica de dois parâmetros de dados sísmicos, a magnitude do corpo da onda x_1 e a magnitude do corpo da superfície x_2, para terremotos (círculos brancos) e explosões nucleares (círculos pretos) ocorridos entre 1982 e 1990 na Ásia e no Oriente Médio (Kebeasy *et al*., 1998). Também é mostrado um limite de decisão entre as classes. (b) O mesmo domínio com mais pontos de dados. Os terremotos e explosões já não são mais linearmente separáveis.

A **fronteira de decisão** é uma linha (ou uma superfície, em dimensões superiores) que separa as duas classes. Na Figura 18.15(a), a fronteira de decisão é uma linha reta. O limite da decisão linear é chamado de **separador linear** e os dados que admitem tal separador são chamados de **linearmente separáveis**. O separador linear, nesse caso, é definido por

$$x_2 = 1{,}7x_1 - 4.9 \text{ ou} - 4{,}9 + 1{,}7x_1 - x_2 = 0.$$

As explosões, que queremos classificar com valor 1, estão à direita dessa linha com valores maiores de x_1 e valores menores de x_2; então eles são pontos para os quais $-4{,}9 + 1{,}7x_1 - x_2 > 0$,

enquanto os terremotos têm $-4,9 + 1,7x_1 - x_2 < 0$. Usando a convenção de entrada fictícia $x_0 = 1$, podemos escrever a hipótese de classificação como

$$h_{\mathbf{w}}(\mathbf{x}) = 1 \text{ se } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ e } 0 \text{ caso contrário.}$$

Alternativamente, podemos pensar em h como o resultado de passar a função linear $\mathbf{w} \cdot \mathbf{x}$ através de uma **função de limiar**:

$$h_{\mathbf{w}}(\mathbf{x}) = Limiar(\mathbf{w} \cdot \mathbf{x}), \text{ onde } Limiar(z) = 1 \text{ se } z \geq 0 \text{ e } 0 \text{ caso contrário.}$$

A função de limiar é mostrada na Figura 18.17(a).

Agora que a hipótese $h_{\mathbf{w}}(\mathbf{x})$ tem forma matemática bem definida, podemos pensar em escolher os pesos $\mathbf{w}$ para minimizar a perda. Nas Seções 18.6.1 e 18.6.2, fizemos isso de forma fechada (definindo o gradiente como zero e resolvendo para os pesos) e por gradiente de descida no espaço de peso. Aqui não podemos fazer nenhuma dessas coisas porque o gradiente é zero em quase todo o espaço de peso, exceto nos pontos em que $\mathbf{w} \cdot \mathbf{x} = 0$ e naqueles pontos em que o gradiente é indefinido.

Há, no entanto, uma regra simples de atualização de peso que converge para uma solução, isto é, um separador linear que classifica os dados perfeitamente desde que os dados sejam linearmente separáveis. Para um único exemplo $(\mathbf{x}, y)$, temos

$$w_i \leftarrow w_i + \alpha \, (y - h_{\mathbf{w}}(\mathbf{x})) \times x_i \qquad (18.7)$$

que é essencialmente idêntico à Equação 18.6, a regra de atualização para a regressão linear! Essa regra é chamada de **regra de aprendizagem perceptron**, por razões que serão esclarecidas na Seção 18.7. Como estamos considerando um problema de classificação 0/1, no entanto, o comportamento é um pouco diferente. Tanto o valor verdadeiro y como a hipótese de saída $h_{\mathbf{w}}(\mathbf{x})$ são 0 ou 1, por isso há três possibilidades:

- Se a saída está correta, ou seja, $y = h_{\mathbf{w}}(\mathbf{x})$, os pesos não são alterados.
- Se y for 1, mas $h_{\mathbf{w}}(\mathbf{x})$ for 0, w_i será *aumentado* quando a entrada correspondente x_i for positiva e diminuído quando x_i for negativo. Isso faz sentido porque queremos fazer $\mathbf{w} \cdot \mathbf{x}$ maior para que $h_{\mathbf{w}}(\mathbf{x})$ gere um 1.
- Se y for 0, mas $h_{\mathbf{w}}(\mathbf{x})$ for 1, w_i será diminuído quando a entrada correspondente x_i for positiva e aumentado quando x_i for negativo. Isso faz sentido porque queremos fazer $\mathbf{w} \cdot \mathbf{x}$ menor para que $h_{\mathbf{w}}(\mathbf{x})$ gere um 0.

Normalmente, a regra de aprendizagem é aplicada em um exemplo de cada vez, escolhendo exemplos ao acaso (como no gradiente estocástico de descida). A Figura 18.16(a) mostra uma **curva de treinamento** para essa regra de aprendizagem aplicada aos dados do terremoto/explosão mostrados na Figura 18.15(a). A curva de treinamento mede o desempenho do classificador em um conjunto de treinamento fixo à medida que o processo de aprendizagem prossegue naquele mesmo

conjunto de treinamento. A curva mostra a regra de atualização convergindo para um separador linear de erro zero. O processo de "convergência" não é exatamente belo, mas sempre funciona. Essa execução particular leva 657 etapas para convergir para um conjunto de dados com 63 exemplos, de modo que cada exemplo seja apresentado cerca de 10 vezes, em média. É previsível que a variação entre as execuções seja muito grande.

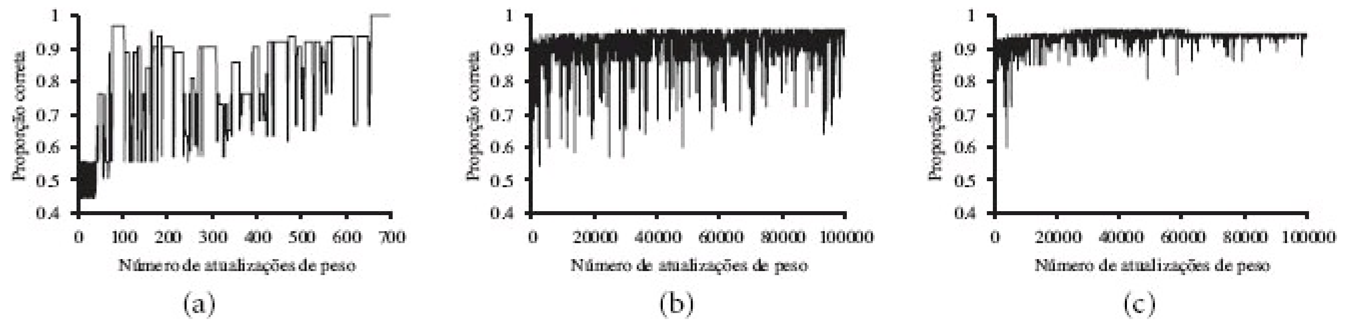


Figura 18.16 (a) Representação gráfica da precisão do conjunto de treinamento total *versus* o número de iterações através do conjunto de treinamento para a regra de aprendizagem perceptron, com base nos dados do terremoto/explosão da Figura 18.15(a). (b) A mesma representação gráfica para os dados não separáveis, ruidosos da Figura 18.15(b); observe a mudança na escala do eixo *x*. (C) A mesma representação gráfica como em (b), com uma programação de taxa de aprendizagem (t) = 1000/(1000 + t).

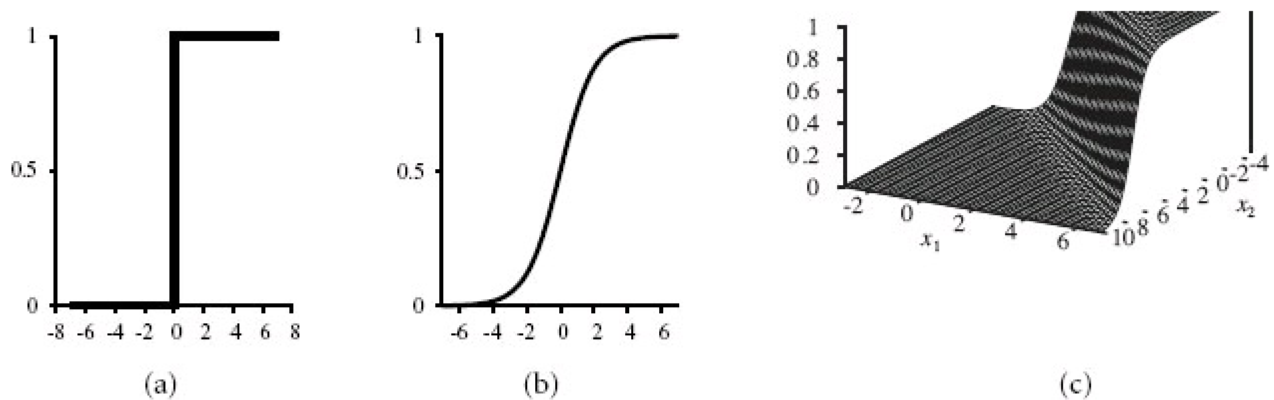


Figura 18.17 (a) A função de limiar rígido *Limiar*(z) com saída 0/1. Observe que a função é não diferençável em $z = 0$. (b) A função logística, *Logística*(z) = $\frac{1}{1+e^{-z}}$ é também conhecida como função sigmoide. (c) Representação gráfica de uma hipótese de regressão logística $h_{\mathbf{w}}(\mathbf{x})$ = *Logística*($\mathbf{w} \cdot \mathbf{x}$) para os dados mostrados na Figura 18.15(b).

Dissemos que a regra de aprendizagem do perceptron converge para um separador linear perfeito quando os pontos de dados são linearmente separáveis, mas, e se não forem? Essa situação é muito comum no mundo real. Por exemplo, a Figura 18.15(b) adiciona de volta os pontos de dados omitidos por Kebeasy *et al*. (1998) quando eles marcaram os dados mostrados na Figura 18.15(a). Na Figura 18.16(b), mostramos que a regra de aprendizagem perceptron não conseguiu convergir, mesmo depois de 10 mil etapas: mesmo que ela atinja a solução de erro mínimo (três erros), muitas vezes o algoritmo continuará mudando os pesos. Em geral, a regra perceptron não pode convergir para uma solução estável para a taxa de aprendizagem fixada a, mas se decai para $O(1/t)$ onde t é o número de iteração, a regra pode ser mostrada para convergir para uma solução de erro mínimo quando os exemplos forem apresentados em uma sequência aleatória.[7] Também se pode mostrar que

encontrar a solução mínima de erros é NP-difícil, por isso espera-se que muitas apresentações de exemplos sejam necessárias para que a convergência seja alcançada. A Figura 18.16(b) mostra o processo de treinamento com uma programação de taxa de aprendizagem α (t) = 1000/(1000 + t): a convergência não é perfeita depois de 100.000 iterações, mas é muito melhor do que o caso fixo α.

18.6.4 Classificação linear com regressão logística

Vimos que passar a saída de uma função linear através da função de limiar cria um classificador linear; também a natureza rígida do limiar causa alguns problemas: a hipótese $h_{\mathbf{w}}(\mathbf{x})$ não é diferenciável, e é na verdade fato uma função descontínua de suas entradas e seus pesos, o que torna a aprendizagem com a regra perceptron uma aventura muito imprevisível. Além disso, o classificador linear sempre anuncia uma previsão completamente confiante de 1 ou 0, mesmo para exemplos que estão muito perto da fronteira; em muitas situações, precisamos realmente de previsões mais graduais.

Todos esses problemas podem ser resolvidos em grande medida suavizando a função de limiar, aproximando o limiar rígido com uma função contínua, diferenciável. No Capítulo 14, vimos duas funções que parecem limiares suaves: integral de distribuição normal padrão (usada para o modelo probit) e a função logística (usada para o modelo logit). Embora as duas funções tenham forma muito semelhante, a função logística

$$Logística(z) = \frac{1}{1 + e^{-z}}$$

tem mais propriedades matemáticas convenientes. A função é mostrada na Figura 18.17(b). Com a função logística substituindo a função de limiar, agora temos

$$h_{\mathbf{w}}(\mathbf{x}) = Logística\ (\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}} .$$

Um exemplo de tal hipótese para o problema de duas entradas terremoto/explosão é mostrado na Figura 18.17(c). Observe que a saída, sendo um número entre 0 e 1, pode ser interpretada como uma *probabilidade* de pertencer à classe rotulada de 1. A hipótese forma um limiar suave no espaço de entrada e dá uma probabilidade de 0,5 para qualquer entrada ao centro da região de limiar, e às abordagens 0 ou 1 à medida que nos afastamos do limiar.

O processo de ajuste dos pesos desse modelo para minimizar a perda em um conjunto de dados é chamado de **regressão logística**. Não há solução fácil de forma fechada para encontrar o valor ótimo de $\mathbf{w}$ com esse modelo, mas o cálculo da descida pelo gradiente é simples. Devido a nossas hipóteses não gerarem apenas 0 ou 1, vamos utilizar a função de perda L_2; para manter as fórmulas legíveis vamos também usar g para representar a função logística, com g' sua derivada.

Para um único exemplo ($\mathbf{x}$, y), a derivação do gradiente é a mesma que para a regressão linear (Equação 18.5) até o ponto onde é inserida a forma real de h. (Para essa derivada, precisaremos da **regra da cadeia**, $\partial g(f(x))/\partial x = g'(f(x))\partial f(x)/\partial x$.)) Temos

$$
\begin{aligned}
\frac{\partial}{\partial w_i} Perda\ (\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\
&= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x})) \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i}\mathbf{w} \cdot \mathbf{x} \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i\ .
\end{aligned}
$$

A derivada g' da função logística satisfaz $g'(z) = g(z)\ (1 - g\ (z))$, então temos

$$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - g(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

assim, a atualização de peso para minimizar a perda é

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i. \qquad (18.8)$$

Repetindo os experimentos da Figura 18.16 com regressão logística em vez de classificador de limiar linear, obtemos os resultados mostrados na Figura 18.18. Em (a), como o caso é separável linearmente, a regressão logística é um pouco mais lenta para convergir, mas tem um comportamento muito mais previsível. Em (b) e (c), onde os dados são ruidosos e não separáveis, a regressão logística converge muito mais rápido e confiavelmente. Essas vantagens tendem a se manter em aplicações do mundo real, e a regressão logística tornou-se uma das técnicas de classificação mais populares para problemas de medicina, marketing e análise de dados, pontuação de crédito, saúde pública e outras aplicações.

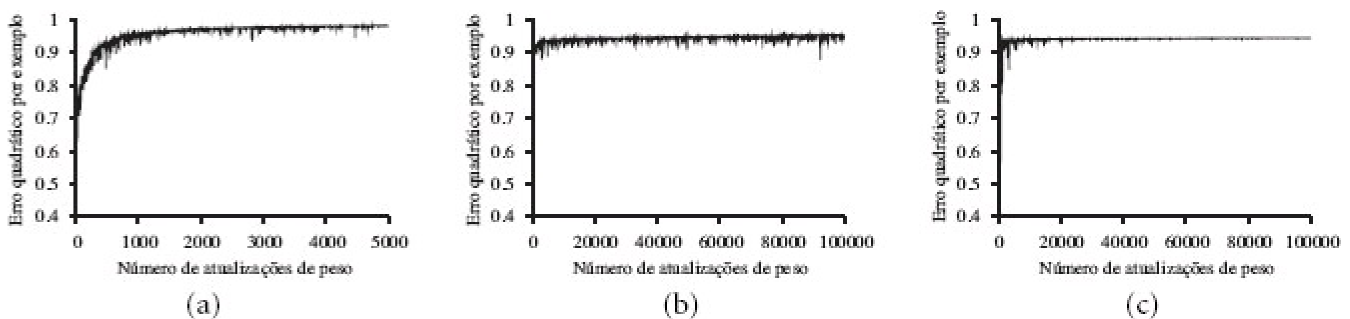


Figura 18.18 Repetição dos experimentos na Figura 18.16 por regressão logística e erro quadrático. A representação gráfica em (a) abrange 5.000 iterações em vez de 1.000, enquanto (b) e (c) utilizam a mesma escala.

18.7 REDES NEURAIS ARTIFICIAIS

Passaremos agora ao que parece ser um tema um tanto independente: o cérebro. Na verdade, como veremos, as ideias técnicas que discutimos até agora neste capítulo serão úteis na construção de modelos matemáticos da atividade do cérebro e, de forma inversa, pensar sobre o cérebro tem ajudado a estender o âmbito das ideias técnicas.

O Capítulo 1 passou brevemente pelos resultados básicos da neurociência — em particular, a hipótese de que a atividade mental consiste basicamente na atividade eletroquímica em redes de

células cerebrais chamadas **neurônios** (a Figura 1.2 mostrou um diagrama esquemático de um neurônio típico). Inspirado por essa hipótese, alguns dos trabalhos mais antigos de IA tiveram o objetivo de criar **redes neurais artificiais** (outros nomes do campo incluem **conexionismo**, **processamento distribuído em paralelo** e **computação neural**). A Figura 18.19 apresenta um modelo matemático simples do neurônio desenvolvido por McCulloch e Pitts (1943). Grosseiramente falando, ele "dispara" quando uma combinação linear de suas entradas excede algum limiar (rígido ou suave), ou seja, ele implementa um classificador linear do tipo descrito na seção anterior. Uma rede neural é apenas uma coleção de unidades conectadas; as propriedades da rede são determinadas pela sua topologia e pelas propriedades dos "neurônios".

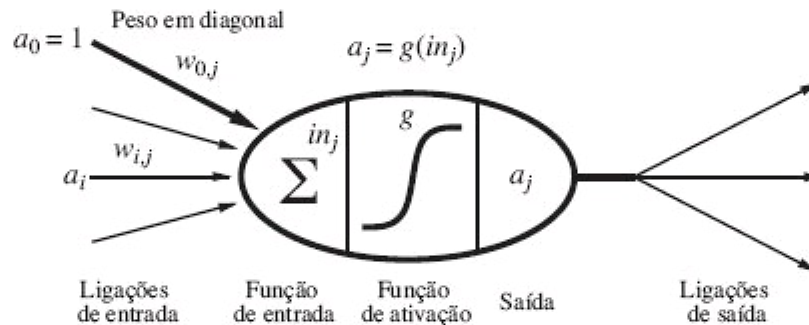


Figura 18.19 Modelo matemático simples de um neurônio. A ativação de saída da unidade é $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, onde a_i é a ativação de saída da unidade i e $w_{i,j}$ é o peso sobre a ligação da unidade i com essa unidade.

Desde 1943, têm sido desenvolvidos modelos muito mais detalhados e realistas, tanto de neurônios como de sistemas maiores no cérebro, levando ao campo moderno da **neurociência computacional**. Por outro lado, os pesquisadores de IA e os estatísticos tornaram-se interessados nas propriedades mais abstratas das redes neurais, tais como sua capacidade de realizar computação distribuída, de tolerar entradas ruidosas e aprender. Embora entendamos agora que outros tipos de sistemas — incluindo redes bayesianas — têm essas propriedades, as redes neurais permanecem uma das formas mais populares e eficazes de aprendizagem do sistema e são dignos de estudo.

18.7.1 Estrutura das redes neurais

As redes neurais são compostas por nós ou **unidades** (ver Figura 18.19) conectadas por **ligações** direcionadas. Uma ligação da unidade i para a unidade j serve para propagar a **ativação** a_i de i para j.[8] Cada ligação também tem um **peso** numérico $w_{i,j}$ associado a ele, que determina a força e o sinal de conexão. Assim como em modelos de regressão linear, cada unidade tem uma entrada fictícia $a_0 = 1$ com peso associado $w_{0,j}$. Cada unidade j primeiro calcula uma soma ponderada de suas entradas:

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i .$$

Em seguida, aplica uma **função de ativação** g a essa soma para obter a saída:

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right) . \qquad (18.9)$$

A ativação da função g tipicamente é tanto um limiar rígido (Figura 18.17(a)), caso em que a unidade é chamada de **perceptron**, como uma função logística (Figura 18.17 (b)), caso em que por vezes é utilizado o termo **perceptron sigmoide**. Ambas as funções de ativação não linear garantem a propriedade importante de que toda a rede de unidades pode representar uma função não linear (veja o Exercício 18.22). Como mencionado na discussão de regressão logística, a função de ativação logística tem a vantagem adicional de ser diferenciável.

Tendo decidido sobre o modelo matemático para "neurônios" individuais, a próxima tarefa é conectá-los para formar uma rede. Existem duas formas fundamentalmente distintas para fazer isso. Uma **rede com alimentação para a frente** tem conexões somente em uma direção, isto é, forma um grafo acíclico dirigido. Cada nó recebe a entrada de nós "para cima" e libera a saída de nós "para baixo"; não há laços. Uma rede com alimentação para a frente representa uma função de sua entrada atual; portanto, não tem estado interno que não seja os próprios pesos. A **rede recorrente**, por outro lado, alimenta suas saídas de volta às suas próprias entradas. Isso significa que os níveis de ativação da rede formam um sistema dinâmico que pode atingir um estado estável ou apresentar oscilações ou até mesmo um comportamento caótico. Além disso, a resposta da rede para determinada entrada depende do seu estado inicial, que pode depender de entradas anteriores. Portanto, as redes recorrentes (ao contrário das redes com alimentação para a frente) podem suportar memória de curto prazo. Isso as torna mais interessantes como modelos de cérebro, mas também mais difícil de entender. Esta seção vai se concentrar em redes com alimentação para a frente; algumas dicas para mais informação sobre redes recorrentes são dadas ao final do capítulo.

As redes com alimentação para a frente normalmente estão dispostas em **camadas**, de tal forma que cada unidade recebe a entrada somente a partir de unidades na camada imediatamente anterior. Nas duas próximas subseções, analisaremos as redes de camada única, em que cada unidade se conecta diretamente a partir de entradas da rede para suas saídas, e as redes de camadas múltiplas, que têm uma ou mais camadas de **unidades ocultas** que não são conectadas às saídas da rede. Até agora, neste capítulo, consideramos apenas problemas de aprendizagem com uma variável única de saída y, mas as redes neurais são frequentemente usadas em casos em que são adequadas saídas múltiplas. Por exemplo, se quisermos formar uma rede para adicionar dois bits de entrada, cada 0 ou 1, precisaremos de uma saída para o bit de soma e uma para o bit de vai-um. Além disso, quando o problema de aprendizagem envolve classificação em mais que duas classes — por exemplo, quando aprendemos a categorizar imagens de dígitos manuscritos —, é comum o uso de uma unidade de saída para cada classe.

18.7.2 Redes neurais de camada única com alimentação para a frente (perceptrons)

Uma rede com todas as entradas conectadas diretamente com as saídas é chamada de **rede neural de camada única** ou **rede perceptron**. A Figura 18.20 mostra uma rede perceptron simples de duas

entradas e duas saídas. Com uma rede desse tipo, podemos ter a esperança de aprender a função adicionador de dois bits, por exemplo. Aqui estão todos os dados de treinamento de que precisaremos:

x_1	x_2	y_3 (transporte)	y_4 (soma)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

A primeira coisa a notar é que uma rede perceptron com *m* saídas é realmente *m* redes separadas, pois cada peso afeta apenas uma das saídas. Assim, haverá *m* processos de treinamento separados. Além disso, dependendo do tipo de função de ativação utilizada, o processo de treinamento será tanto a **regra de aprendizagem perceptron** (Equação 18.7) como a regra de descida pelo gradiente por **regressão logística** (Equação 18.8).

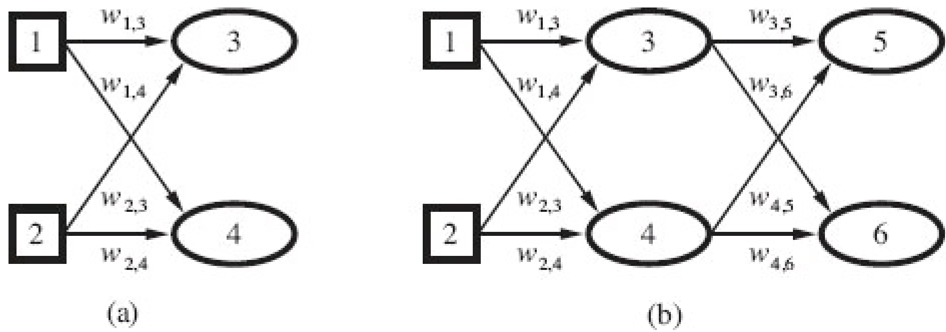


Figura 18.20 (a) Rede perceptron com duas unidades de entrada e duas unidades de saída. (b) Rede neural com duas entradas, uma camada oculta de duas unidades e uma unidade de saída. As entradas fictícias e seus pesos associados não são mostrados.

Se você tentar qualquer método nos dados somadores de dois bits, algo interessante acontecerá. A unidade 3 aprende a função vai-um com facilidade, mas a unidade 4 falha completamente em aprender a função soma. Não, a unidade 4 não está com defeito! O problema é com a função de soma em si. Vimos na Seção 18.6 que classificadores lineares (rígidos ou suaves) podem representar limiares de decisão linear no espaço de entrada. Isso funciona bem para a função vai-um, que é uma lógica E (ver Figura 18.21(a)). A função soma, porém, é um XOR (OU exclusivo) das duas entradas. Como a Figura 18.21(c) ilustra, essa função não é linearmente separável de modo que o perceptron não pode aprendê-la.

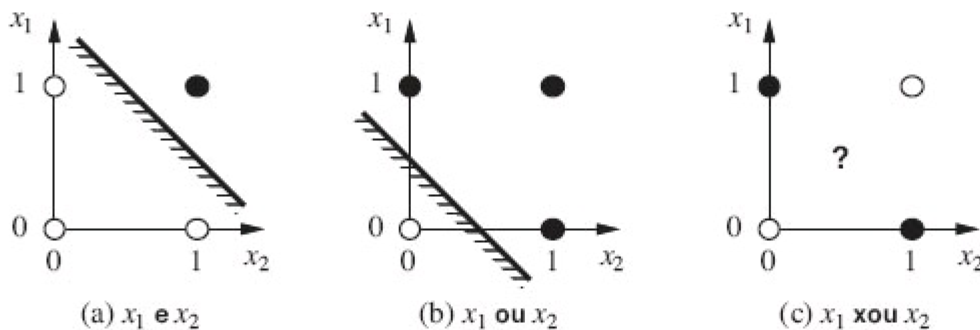


Figura 18.21 Separabilidade linear em limiar de perceptrons. Os pontos pretos indicam um ponto no espaço de entrada onde o valor da função é 1, e pontos brancos indicam um ponto onde o valor é 0. O perceptron retorna 1 na região do lado não sombreado da linha. Em (c), não existe essa linha que classifica corretamente as entradas.

As funções linearmente separáveis constituem apenas uma pequena fração de todas as funções booleanas; o Exercício 18.20 pede para quantificar essa fração. A incapacidade dos perceptrons de aprender mesmo tais funções simples como XOR foi um revés significativo para a comunidade nascente de redes neurais na década de 1960. No entanto, os perceptrons estão longe de ser inúteis. A Seção 18.6.4 observou que a regressão logística (ou seja, o treinamento de um perceptron sigmoide) ainda é hoje muito popular e uma ferramenta eficaz. Além disso, um perceptron pode representar uma função booleana bastante "complexa" de forma compacta. Por exemplo, a **função da maioria**, que gera um 1 apenas se mais da metade de suas entradas *n* forem 1, pode ser representada por um perceptron com cada $w_i = 1$ e com $w_0 = -n/2$. Uma árvore de decisão precisaria exponencialmente de muitos mais nós para representar essa função.

A Figura 18.22 mostra a curva de aprendizagem para um perceptron em dois problemas diferentes. À esquerda, mostramos a curva de aprendizagem da função maioria com 11 entradas booleanas (ou seja, a função de saída 1, se seis ou mais entradas forem 1). Como seria de esperar, o perceptron aprende a função de forma rápida porque a função da maioria é linearmente separável. Por outro lado, o aprendiz de árvore de decisão não faz nenhum progresso porque a função da maioria é muito difícil (embora não seja impossível) para ser representada como árvore de decisão. À direita, temos o exemplo do restaurante. A solução do problema é facilmente representada como uma árvore de decisão, mas não é linearmente separável. O melhor plano utilizando dados classifica corretamente apenas 65%.

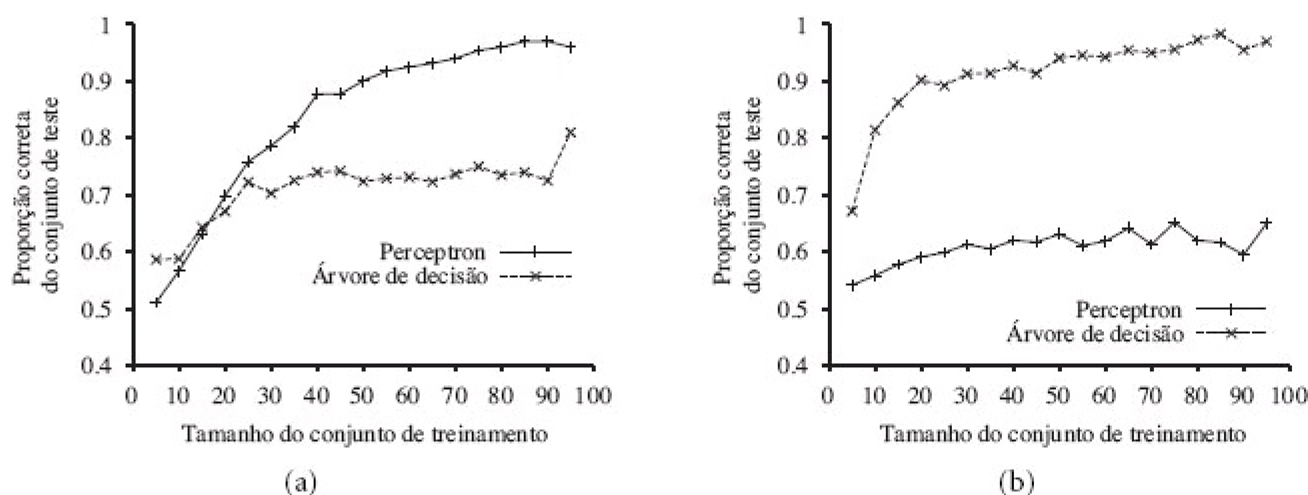


Figura 18.22 Comparação do desempenho dos perceptrons e das árvores de decisão. (a) Os perceptrons são melhores em aprender a função de maioria de 11 entradas. (b) As árvores de decisão são melhores em aprender o predicado *VamosEsperar* no exemplo do restaurante.

18.7.3 Redes neurais com alimentação para a frente multicamada

McCulloch e Pitts (1943) sabiam que uma única unidade de limiar não resolveria todos os seus

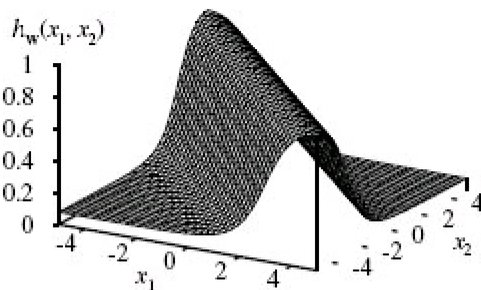
problemas. De fato, suas teses provam que tal unidade pode representar funções booleanas básicas E, OU e NÃO, e depois continuam a argumentar que qualquer funcionalidade desejada pode ser obtida ligando grande número de unidades em redes de profundidade arbitrária (possivelmente recorrentes). O problema era que ninguém sabia como treinar tais redes.

Isso acaba por ser um problema fácil se pensarmos em uma rede da maneira certa: como uma função $h_{\mathbf{w}}(\mathbf{x})$ parametrizada pelos pesos **w**. Considere a rede simples mostrada na Figura 18.20 (b), que tem duas unidades de entrada, duas unidades ocultas e duas unidades de saída (além disso, cada unidade tem uma entrada fictícia fixada em 1). Dado um vetor de entrada $\mathbf{x} = (x_1, x_2)$, as ativações das unidades de entrada são definidas como $(a_1, a_2) = (x_1, x_2)$. A saída da unidade 5 é dada por

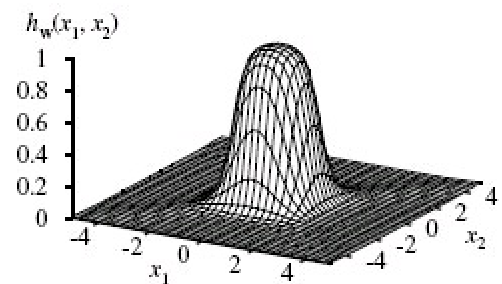
$$\begin{aligned} a_5 &= g(w_{0,5,+}w_{3,5}\, a_3 + w_{4,5}\, a_4) \\ &= g(w_{0,5,+}w_{3,5}\, g(w_{0,3} + w_{1,3}\, a_1 + w_{2,3}\, a_2) + w_{4,5}\, g(w_0 4 + w_{1,4}\, a_1 + w_{2,4}\, a_2)) \\ &= g(w_{0,5,+}w_{3,5}\, g(w_{0,3} + w_{1,3}\, x_1 + w_{2,3}\, x_2) + w_{4,5}\, g(w_0 4 + w_{1,4}\, x_1 + w_{2,4}\, x_2)). \end{aligned}$$

Assim, temos a saída expressa como uma função das entradas e dos pesos. Uma expressão similar vale para a unidade 6. Enquanto podemos calcular as derivadas de tais expressões com relação aos pesos, podemos usar o método de minimização de perda da descida pelo gradiente para treinar a rede. A Seção 18.7.4 mostra exatamente como fazer isso. E, como a função representada por uma rede pode ser altamente não linear — composta por funções de limiar suave não lineares aninhadas — pode-se ver as redes neurais como ferramenta para fazer **regressão não linear**.

Antes de nos aprofundar em regras de aprendizagem, vamos observar como as redes geram funções complicadas. Primeiro, lembre-se de que cada unidade em uma rede sigmoide representa um limiar suave em seu espaço de entrada, como mostrado na Figura 18.17(c). Com uma camada oculta e uma camada de saída, como na Figura 18.20(b), cada unidade de saída calcula uma combinação linear de limiar suave dessas várias funções. Por exemplo, pela adição de duas funções de limiar suave opostas e limitando o resultado, podemos obter uma função "cume" como mostrado na Figura 18.23(a). A combinação das duas cristas em ângulos corretos entre si (ou seja, combinando as saídas de quatro unidades ocultas), obtém um "sobressalto", como mostrado na Figura 18.23(b).



(a)



(b)

Figura 18.23 (a) Resultado da combinação de duas funções de limiar suave opostas para produzir um cume. (b) Rresultado da combinação de duas cristas para produzir um sobressalto.

Com mais unidades ocultas, podemos produzir outros sobressaltos de tamanhos diferentes em outros lugares. Na verdade, com uma camada oculta única, suficientemente grande, é possível representar qualquer função contínua de entrada com precisão arbitrária; com duas camadas, podem

ser representadas até mesmo funções descontínuas.[9] Infelizmente, para qualquer estrutura de rede *particular*, é mais difícil caracterizar exatamente as funções que podem ser representadas e as que não podem.

18.7.4 Aprendizagem em redes multicamadas

Primeiro, vamos tratar de uma complicação menor decorrente de redes multicamadas: interações entre os problemas de aprendizagem quando a rede tem várias saídas. Nesses casos, devemos pensar na rede como uma função vetorial $\mathbf{h_w}$ de implementação, em vez de uma função escalar $\mathbf{h_w}$; por exemplo, a rede na Figura 18.20(b) retorna um vetor $[a_5, a_6]$. Da mesma forma, a saída de destino será um vetor $\mathbf{y}$. Considerando que uma rede perceptron se decompõe em m problemas de aprendizagem em separado para um problema de m saídas, essa decomposição falha em uma rede de múltiplas camadas. Por exemplo, tanto a_5 como a_6 na Figura 18.20(b) dependem de todos os pesos da camada de entrada, de modo que as atualizações desses pesos dependerão de erros tanto em a_5 como em a_6. Felizmente, essa dependência é muito simples, no caso de qualquer função de perda que seja *aditiva* entre os componentes do vetor de erro $\mathbf{y} - \mathbf{h_w}(\mathbf{x})$. Pela perda L_2, temos, para qualquer peso w,

$$\frac{\partial}{\partial w} \, Perda\,(\mathbf{w}) = \frac{\partial}{\partial w}|\mathbf{y} - \mathbf{h_w}(\mathbf{x})|^2 = \frac{\partial}{\partial w}\sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial w}(y_k - a_k)^2 \qquad (18.10)$$

onde o índice k varia no intervalo dos nós na camada de saída. Cada termo, no somatório final, é apenas o gradiente de perda para a k-ésima saída, calculado como se outras saídas não existissem. Assim, podemos decompor um problema de aprendizagem com m-saídas em m problemas de aprendizagem, desde que não esqueçamos de somar as contribuições do gradiente de cada um deles ao atualizar os pesos.

A principal complicação vem da adição de camadas ocultas da rede. Enquanto que o erro $\mathbf{y} - \mathbf{h_w}$ na camada de saída é claro, o erro nas camadas ocultas parece misterioso porque os dados de treinamento não dizem que valor os nós ocultos devem ter. Felizmente, verifica-se que podemos **retropropagar** o erro da camada de saída para as camadas ocultas. O processo de retropropagação emerge diretamente de uma derivação do gradiente de erro geral. Primeiro, descreveremos o processo com uma justificação intuitiva; depois, mostraremos a derivação.

Na camada de saída, a regra de atualização do peso é idêntica à Equação 18.8. Temos unidades de saída múltiplas; assim, façamos Err_k o componente do erro k-ésimo do vetor de erro $\mathbf{y} - \mathbf{h_w}$. Também é conveniente definir um erro modificado $\Delta_k = Err_k \times g'(in_k)$, para que a regra de atualização de peso torne-se

$$w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k \,. \qquad (18.11)$$

Para atualizar as conexões entre as unidades de entrada e as unidades ocultas, precisamos definir uma quantidade análoga ao termo de erro dos nós de saída. Aqui fazemos a retropropagação do erro.

A ideia é que o nó oculto j seja "responsável" por uma fração do erro Δ_k em cada um dos nós de saída aos quais ele se conecta. Assim, os valores Δ_k são divididos de acordo com a força de ligação entre o nó oculto e o nó de saída e são retropropagados para fornecer os valores Δ_j para a camada oculta. A regra para a propagação dos valores Δ é a seguinte:

$$\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k \ . \qquad (18.12)$$

Agora a regra de atualização de peso para os pesos entre as entradas e a camada oculta é essencialmente idêntica à regra de atualização para a camada de saída:

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j \ .$$

O processo de retropropagação pode ser resumido da seguinte forma:

- Calcule valores Δ para as unidades de saída usando o erro observado.
- A partir da camada de saída, repita o seguinte para cada camada da rede até que a primeira camada oculta seja alcançada:
  - Propague os valores Δ de volta à camada anterior.
  - Atualize os pesos entre as duas camadas.

O algoritmo detalhado é mostrado na Figura 18.24.

```
função APRENDIZAGEM-DE-RETRO-PROP(exemplos, rede) retorna uma rede neural
   entradas: exemplos, um conjunto de exemplos, cada um com vetor de entrada x e vetor de saída y
             rede, uma rede multicamadas com L camadas, pesos w_ij e unção de ativação g
   variáveis locais: Δ, um vetor de erros, indexado pelo nó de rede

   repetir
      para cada peso w_i,j na rede faça
         w_i,j ← um número randômico pequeno
      para cada exemplo (x, y) em exemplos faça
         /* Propagar as entradas para a frente para computar as saídas */
         para cada nó i na camada de entrada faça
            a_i ← x_i
         para ℓ = 2 até L faça
         para cada nó j na camada ℓ faça
            in_j ← Σ_i w_i,j a_i
            a_j ← g(in_j)
         /* Propagar deltas retrocedendo da camada de saída para a camada de entrada */
         para cada nó j na camada de saída faça
            Δ[j] ← g′(in_j) × (y_j − a_j)
         para ℓ = L − 1 até 1 faça
            para cada nó i na camada ℓ faça
               Δ[i] ← g′(in_i) Σ_j w_i,j Δ[j]
         /* Atualiza cada peso na rede usando deltas */
         para cada peso wi,j na rede faça
            w_i,j ← w_i,j + α × a_i × Δ[j]
   até que algum critério de parada seja satisfeito
   retorno rede
```

Figura 18.24 Algoritmo de retropropagação para o aprendizagem em redes multicamadas.

Para quem gosta de matemática, vamos agora derivar as equações de retropropagação dos primeiros princípios. A derivação é bastante semelhante ao cálculo de gradiente de regressão logística (que conduziu à Equação 18.8), exceto que temos de usar a regra de cadeia mais de uma vez.

A partir da Equação 18.10, calculamos apenas o gradiente de $Perda_k = (y_k - a_k)^2$ na k-ésima saída. O gradiente dessa perda com relação aos pesos de conexão com a camada oculta até a camada de saída será zero, exceto para os pesos $w_{j,k}$ que se unem à k-ésima unidade de saída.

Para aqueles pesos, temos

$$\begin{aligned}
\frac{\partial\, Perda_k}{\partial w_{j,k}} &= -2(y_k - a_k)\frac{\partial a_k}{\partial w_{j,k}} = -2(y_k - a_k)\frac{\partial g(in_k)}{\partial w_{j,k}} \\
&= -2(y_k - a_k)g'(in_k)\frac{\partial in_k}{\partial w_{j,k}} = -2(y_k - a_k)g'(in_k)\frac{\partial}{\partial w_{j,k}}\left(\sum_j w_{j,k}a_j\right) \\
&= -2(y_k - a_k)g'(in_k)a_j = -a_j\Delta_k\ ,
\end{aligned}$$

com Δ_k definido como antes. Para obter o gradiente com relação aos pesos $w_{i,j}$ em conexão com a camada de entrada até a_j camada oculta temos que expandir as ativações a_j e reaplicar a regra da cadeia. Vamos mostrar a derivação em pormenores porque é interessante ver como o operador de derivada retropropaga através da rede:

$$\begin{aligned}
\frac{\partial\, Perda_k}{\partial w_{i,j}} &= -2(y_k - a_k)\frac{\partial a_k}{\partial w_{i,j}} = -2(y_k - a_k)\frac{\partial g(in_k)}{\partial w_{i,j}} \\
&= -2(y_k - a_k)g'(in_k)\frac{\partial in_k}{\partial w_{i,j}} = -2\Delta_k\frac{\partial}{\partial w_{i,j}}\left(\sum_j w_{j,k}a_j\right) \\
&= -2\Delta_k w_{j,k}\frac{\partial a_j}{\partial w_{i,j}} = -2\Delta_k w_{j,k}\frac{\partial g(in_j)}{\partial w_{i,j}} \\
&= -2\Delta_k w_{j,k}g'(in_j)\frac{\partial in_j}{\partial w_{i,j}} \\
&= -2\Delta_k w_{j,k}g'(in_j)\frac{\partial}{\partial w_{i,j}}\left(\sum_i w_{i,j}a_i\right) \\
&= -2\Delta_k w_{j,k}g'(in_j)a_i = -a_i\Delta_j\ ,
\end{aligned}$$

onde Δ_j foi definido como antes. Assim, obtemos as regras de atualização obtidas anteriormente a partir de considerações intuitivas. Também fica claro que o processo pode ser continuado para redes com mais de uma camada oculta, o que justifica o algoritmo geral dado na Figura 18.24.

Passando por toda essa matemática, vamos ver o desempenho de uma rede de camada oculta única no problema do restaurante. Primeiro, precisamos determinar a estrutura da rede. Temos 10 atributos que descrevem cada exemplo, então precisaremos de 10 unidades de entrada. Deveríamos ter uma camada oculta ou duas? Quantos nós em cada camada? Devem estar totalmente conectados? Não há nenhuma boa teoria que nos forneça a resposta (veja a próxima seção). Como sempre, podemos usar a validação cruzada: tentar várias estruturas diferentes e ver qual funciona melhor. Acontece que uma

rede com uma camada oculta contendo quatro nós parece razoável para esse problema. Na Figura 18.25, apresentamos duas curvas. A primeira é uma curva de treinamento mostrando o erro quadrático médio, dado um conjunto de treinamento de 100 exemplos de restaurante durante o processo de atualização de peso. Isso demonstra que a rede, de fato, converge para um ajuste perfeito aos dados de treinamento. A segunda curva é a curva de aprendizagem padrão para os dados do restaurante. A rede neural aprende bem, embora não tão rápido quanto a aprendizagem em árvore de decisão, o que não é surpreendente, em primeiro lugar porque os dados foram gerados a partir de uma árvore de decisão simples.

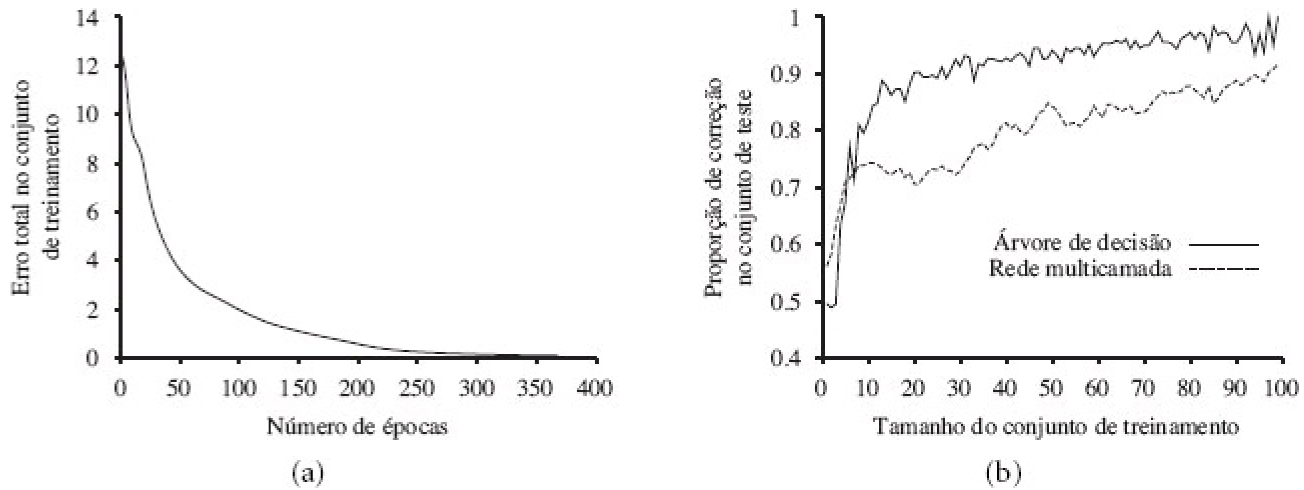


Figura 18.25 (a) A curva de treinamento mostra a redução gradual do erro à medida que os pesos são modificados ao longo de diversas épocas, para um conjunto de exemplos determinados do domínio do restaurante. (b) Curvas de aprendizagem comparativas mostrando que a aprendizagem em árvore de decisão é um pouco melhor para o problema do restaurante do que a retropropagação em uma rede de múltiplas camadas.

Certamente as redes neurais são capazes de tarefas muito mais complexas de aprendizagem, embora deva ser dito que é necessário certa quantidade de esforço para obter a estrutura da rede correta e alcançar a convergência para algo próximo ao ótimo global no espaço de peso. Há literalmente dezenas de milhares de aplicações publicadas de redes neurais. A Seção 18.11.1 apresentará um aplicativo, com mais profundidade.

18.7.5 Aprendendo as estruturas de redes neurais

Até agora, consideramos o problema de pesos de aprendizagem, dada uma estrutura de rede fixa; assim como com redes bayesianas, também precisamos entender como encontrar a melhor estrutura de rede. Se escolhermos uma rede muito grande, ela será capaz de memorizar todos os exemplos, formando uma tabela grande de pesquisa, mas não generalizará bem necessariamente as entradas que nunca foram vistas antes.[10] Em outras palavras, como todos os modelos estatísticos, as redes neurais estão sujeitas a **superadaptações** quando existem muitos parâmetros no modelo. Vimos isso na Figura 18.1, onde os modelos de muitos parâmetros em (b) e (c) ajustam todos os dados, mas podem não generalizar, assim como os modelos de poucos parâmetros em (a) e (d).

Se nos ativermos a redes totalmente conectadas, as únicas escolhas a serem feitas dizem respeito

ao número de camadas ocultas e seus tamanhos. A abordagem usual é tentar várias mantendo as melhores. Será necessária a técnica de **validação cruzad**a se quisermos evitar **espreitar** o conjunto de teste. Ou seja, escolhemos a arquitetura de rede que oferece a maior previsão de precisão nos conjuntos de validação.

Se quisermos considerar redes que não estejam totalmente conectadas, precisamos encontrar algum método de busca eficaz através do espaço muito grande de topologias possíveis de conexão. O **algoritmo de dano cerebral ótimo** começa com uma rede totalmente conectada e remove as conexões dela. Depois que a rede é instruída pela primeira vez, uma abordagem teórica de informação identifica uma seleção ideal de conexões que podem ser descartadas. A rede é, então, reciclada, e se seu desempenho não diminuir, o processo será repetido. Além de remover as conexões, também é possível remover as unidades que não estão contribuindo muito para o resultado.

Vários algoritmos têm sido propostos para produzir uma rede maior de uma menor. Um deles, o algoritmo de ladrilhamento, assemelha-se à aprendizagem com lista de decisão. A ideia é iniciar com uma unidade única que faz o melhor para produzir a saída correta para tantos exemplos de treinamento quanto possível. As unidades subsequentes são adicionadas para cuidar dos exemplos que a primeira unidade obteve errado. O algoritmo adiciona apenas tantas unidades quantas forem necessárias para abranger todos os exemplos.

18.8 MODELOS NÃO PARAMÉTRICOS

A regressão linear e as redes neurais utilizam dados de treinamento para estimar um conjunto fixo de parâmetros **w**. Isso define a nossa hipótese $h_{\mathbf{w}}(\mathbf{x})$, e nesse ponto podemos jogar fora os dados de treinamento porque todos eles estão resumidos por **w**. Um modelo de aprendizagem que resume os dados com um conjunto de parâmetros de tamanho fixo (independentemente do número de exemplos de treinamento) é chamado de **modelo paramétrico**.

Não importa a quantidade de dados que você jogue em um modelo paramétrico, não muda a ideia sobre quantos parâmetros são necessários. Quando os conjuntos de dados são pequenos, faz sentido ter uma restrição forte nas hipóteses permitidas, para evitar a superadaptação. Mas, quando há milhares ou milhões ou bilhões de exemplos para aprender, parece que uma ideia melhor é permitir que os dados falem por si em vez de forçá-los a falar através de um vetor de parâmetros minúsculos. Se os dados informam que a resposta correta é uma função muito sinuosa, não devemos nos restringir às funções lineares ou às ligeiramente sinuosas.

Um **modelo paramétrico** é aquele que não pode ser caracterizado por um conjunto limitado de parâmetros. Por exemplo, suponha que cada hipótese que geramos simplesmente mantenha dentro de si todos os exemplos de treinamento e os use para prever o próximo exemplo. Tal grupo de hipótese será não paramétrico, pois o número efetivo de parâmetros é ilimitado — cresce com o número de exemplos. Essa abordagem é chamada de **aprendizagem baseada em exemplos** ou **aprendizagem baseada em memória**. O método mais simples de aprendizagem baseada em exemplo é a **pesquisa em tabelas**: tome todos os exemplos de treinamento, coloque-os em uma tabela e, depois, quando $h(\mathbf{x})$ for solicitado, veja se x está na tabela; se estiver, devolva o y correspondente. O problema com esse método é que ele não generaliza bem: quando **x** não está na tabela, tudo o que faz é retornar

algum valor default.

18.8.1 Modelo do vizinho mais próximo

Podemos melhorar pesquisa em tabela com uma ligeira variação: dada uma consulta $\mathbf{x}_q$, encontre k exemplos que estiverem mais próximas de $\mathbf{x}_q$. Isso é chamado de pesquisa de k-**vizinhos mais próximos**. Usaremos a notação $VP(k, \mathbf{x}_q)$ para indicar o conjunto de k vizinhos mais próximos.

Para classificar, primeiro encontre $VP(k, \mathbf{x}_q)$ e tome o voto da maioria dos vizinhos (que é o voto majoritário, em caso de classificação binária). Para evitar empates, k é sempre escolhido como número ímpar. Para fazer regressão, podemos tirar a média ou mediana de k vizinhos, ou podemos resolver um problema de regressão linear sobre os vizinhos.

Na Figura 18.26, mostramos o limite de decisão da classificação de k-vizinhos mais próximo para k = 1 e 5 sobre o conjunto de dados do terremoto da Figura 18.15. Os métodos não paramétricos estão sujeitos ainda a subadaptação e superadaptação, assim como os métodos paramétricos. Nesae caso, 1-vizinho mais próximo é a superadaptação; ele reage muito ao preto que está à parte no canto superior direito e ao branco que está à parte em (5,4; 3,7). O limiar de 5-vizinhos mais próximos é bom; valores maiores de k levariam à subadaptação. Como de costume, pode-se usar a validação cruzada para selecionar o melhor valor de k.

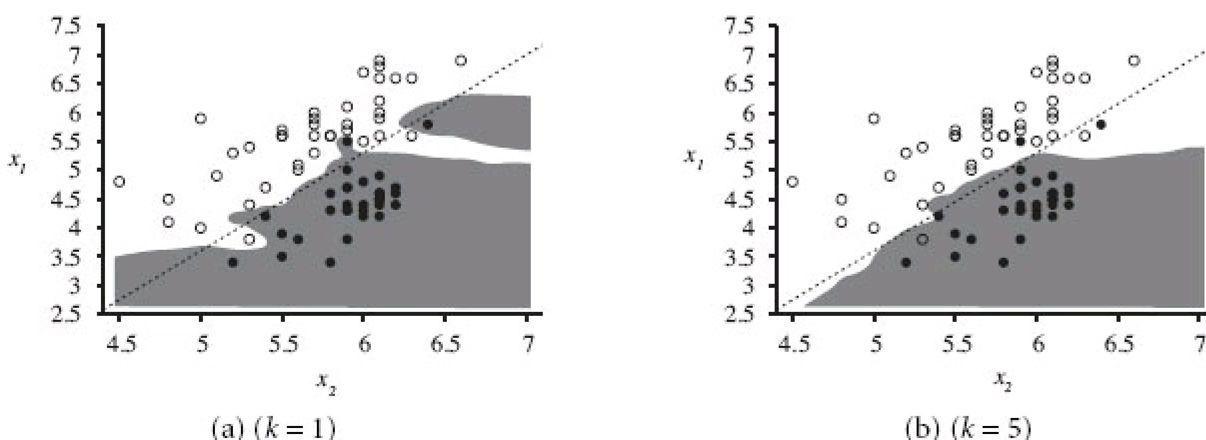


Figura 18.26 (a) Um modelo de k-vizinhos mais próximos mostra a extensão da classe de explosão para os dados da Figura 18.15, com k = 1. A superadaptação é aparente. (b) Com k = 5, o problema de superadaptação desaparece para esse conjunto de dados.

A própria expressão "mais próximo" implica uma métrica de distância. Como podemos medir a distância a partir de um ponto de consulta $\mathbf{x}_q$ até um ponto de exemplo $\mathbf{x}_j$? Prevê-se que as distâncias sejam medidas com a distância de Minkowski ou com a norma L^p, definida como

$$L^p(\mathbf{x}_j, \mathbf{x}_q) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{1/p} .$$

Com p = 2, essa é a distância euclidiana e com p = 1 é a distância de Manhattan. Com valores de atributo booleanos, o número de atributos em que dois pontos diferentes diferem é chamado de

distância de Hamming. Muitas vezes, $p = 2$ é usado se as dimensões estiverem medindo propriedades similares, tais como largura, altura e profundidade de peças em uma correia transportadora, e a distância de Manhattan é utilizada se elas forem diferentes, tais como idade, peso e sexo do paciente. Observe que, se usarmos os números brutos de cada dimensão, a distância total será afetada por uma mudança na escala em qualquer dimensão. Se mudarmos a dimensão i de centímetros para milhas, mantendo as outras dimensões, teremos vizinhos mais próximos diferentes. Para evitar isso, é comum aplicar a **normalização** para medições em cada dimensão. Uma abordagem simples é calcular a média μ_i e o desvio-padrão σ_i dos valores em cada dimensão, e redimensioná-los para que $x_{j,i}$ torne-se $(x_{j,i} - \mu_i)/\sigma_i$. Uma métrica mais complexa conhecida como **distância de Mahalanobis** leva em conta a covariância entre as dimensões.

Em espaços de dimensão baixa com abundância de dados, os vizinhos mais próximos funcionam muito bem: podemos ter dados suficientes nas proximidades para obter uma boa resposta. Mas, à medida que o número de dimensões cresce, deparamo-nos com um problema: os vizinhos mais próximos em espaços dimensionais altos geralmente não estão muito próximos! Considere k-vizinhos mais próximos em um conjunto de dados de N pontos uniformemente distribuídos por todo o interior de uma unidade de hipercubo n-dimensional. Vamos definir a k-vizinhança de um ponto como o menor hipercubo que contém os k-vizinhos mais próximos. Seja l o comprimento médio do lado médio de uma vizinhança. Então, o volume da vizinhança (que contém k pontos) será l^n e o volume do cubo completo (que contém N pontos) será 1. Então, em média, $l^n = k/N$. Extraindo as raízes enésimas de ambos os lados teremos $l = (k/N)^{1/n}$.

Para ser específico, faça $k = 10$ e $N = 1.000.000$. Em duas dimensões ($n = 2$; um quadrado unitário), a vizinhança média tem $l = 0{,}003$, uma pequena fração do quadrado unitário e, em três dimensões, l será apenas 2% do comprimento da aresta do cubo unitário. Já com 17 dimensões, l será metade do comprimento da borda do hipercubo unitário e, em 200 dimensões, será 94%. Esse problema tem sido denominado **maldição da dimensionalidade**.

Outra forma de ver isso: considere os pontos que caem dentro de uma borda compondo 1% do exterior da unidade do hipercubo. Em geral será difícil encontrar um bom valor para eles porque estaremos extrapolando, em vez de interpolando. Em uma dimensão, esses pontos são apenas 2% dos pontos na linha da unidade (aqueles pontos onde $x < 0{,}01$ ou $x > 0{,}99$), mas, em 200 dimensões, mais de 98% dos pontos caem nessa borda fina. Você pode ver um exemplo de ajuste fraco de vizinhos mais próximos na Figura 18.28(b).

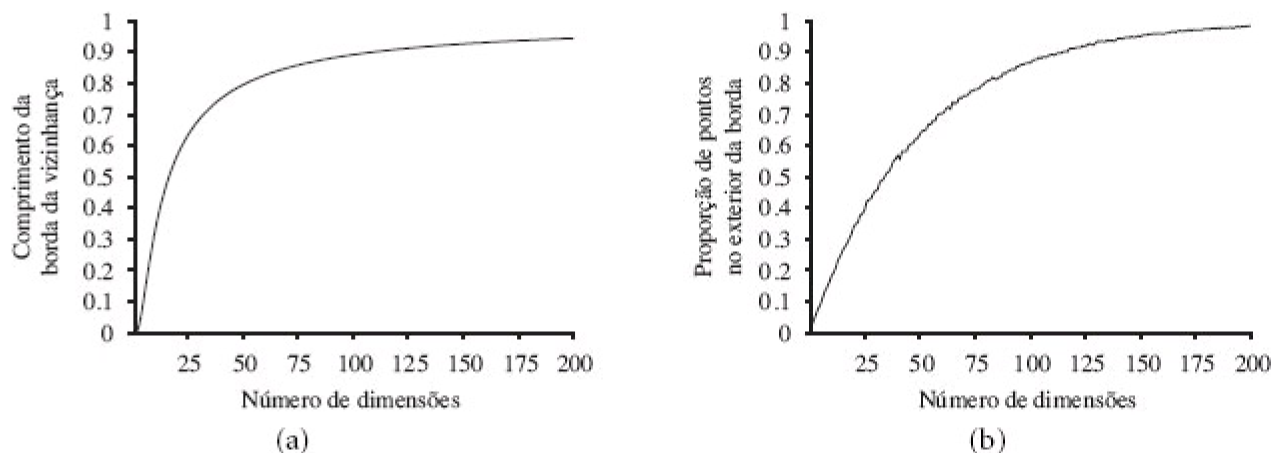


Figura 18.27 A maldição da dimensionalidade. (a) O comprimento da vizinhança média de 10-

vizinhos-mais-próximos em uma unidade de hipercubo com 1.000.000 de pontos, em função do número de dimensões. (b) A proporção de pontos que caem dentro de uma borda fina que consiste em 1% à parte do hipercubo, em função do número de dimensões. Amostragem de 10.000 pontos distribuídos aleatoriamente.

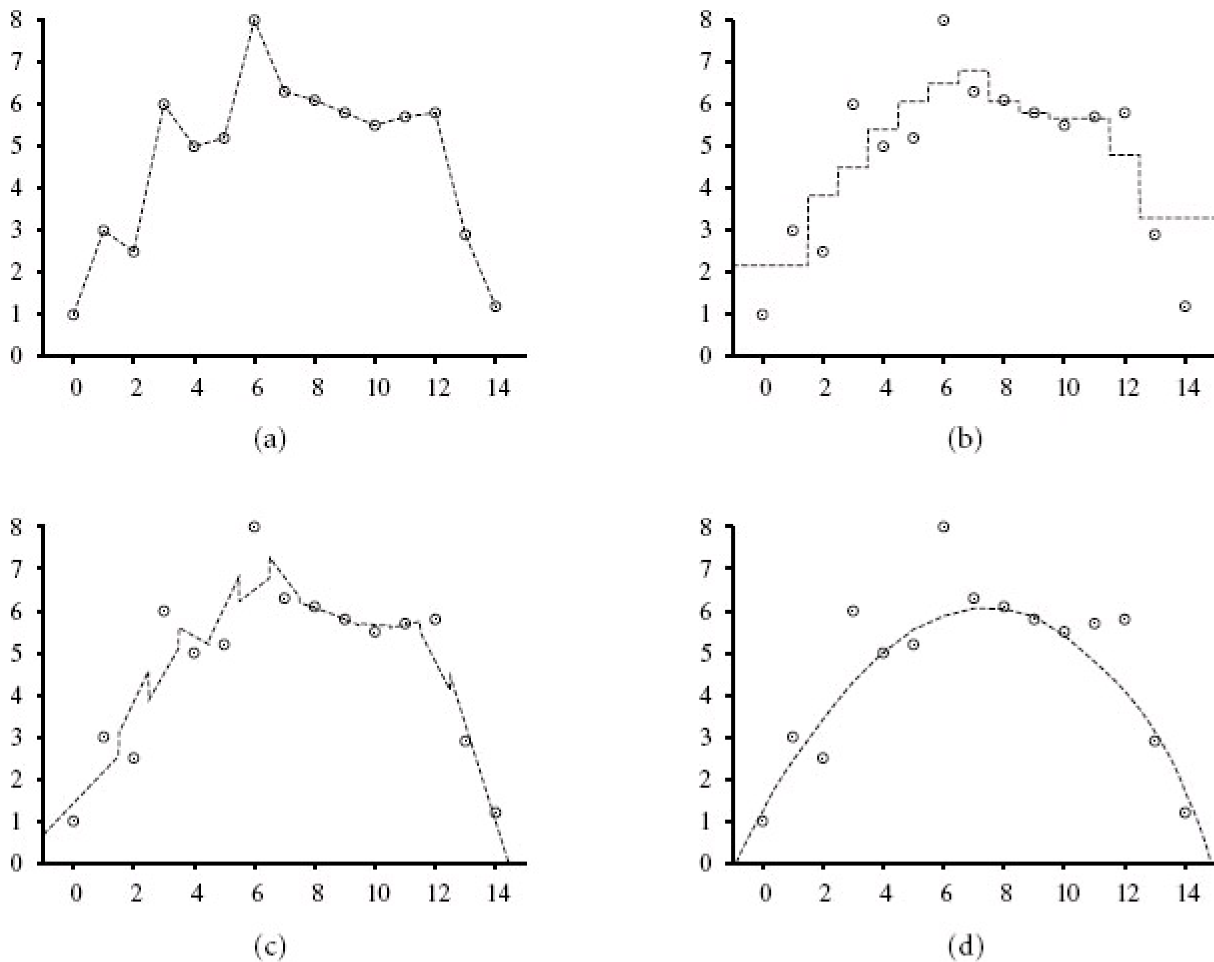


Figura 18.28 Modelos de regressão não paramétrica: (a) ligar os pontos, (b) média dos três vizinhos mais próximos, (c) regressão linear de três vizinhos mais próximos, (d) regressão ponderada localmente com um kernel quadrático de largura $k = 10$.

A função $VP(k, \mathbf{x}_q)$ é simples conceitualmente: dado um conjunto de N exemplos e uma consulta $\mathbf{x}_q$, percorra os exemplos, meça a distância até $\mathbf{x}_q$ de cada um e use os k melhores. Se estiver satisfeito com uma implementação que tem tempo de execução $O(N)$, a história termina aí. Mas os métodos baseados em exemplo são projetados para grandes conjuntos de dados, então ficaríamos satisfeitos com um algoritmo com tempo de execução sublinear. A análise elementar de algoritmos nos diz que a tabela de pesquisa exata é $O(N)$ com uma tabela sequencial, $O(\log N)$ com uma árvore binária e $O(1)$ com uma tabela hash. Agora veremos que árvores binárias e tabelas hash também são aplicáveis para encontrar vizinhos mais próximos.

18.8.2 Encontrar os vizinhos mais próximos com árvores k-d

Uma árvore binária equilibrada sobre os dados com número arbitrário de dimensões é chamada de **árvore k-d**, ou seja árvore k-dimensional (nessa notação, o número de dimensões é n, de modo que

seriam n-d árvores). A construção de uma árvore k-d é similar à construção de uma árvore binária balanceada unidimensional. Começamos com um conjunto de exemplos e no nó raiz dividimo-los ao longo da i-ésima dimensão testando se $x_i \leq m$. Escolhemos o valor m para ser a mediana dos exemplos ao longo da i-ésima dimensão; assim, metade dos exemplos estará na ramificação esquerda da árvore e a outra metade à direita. Em seguida, compomos uma árvore recursivamente para o conjunto de exemplos à esquerda e à direita, parando quando houver menos que dois exemplos à esquerda. Para escolher uma dimensão para dividir cada nó da árvore, pode-se simplesmente selecionar a dimensão i mod n no nível i da árvore (observe que talvez seja necessário dividir várias vezes em qualquer dimensão à medida que descemos na árvore). Outra estratégia é dividir a dimensão que tiver o maior espalhamento de valores.

Pesquisa exata de uma árvore k-d é o mesmo que examinar uma árvore binária (com uma ligeira complicação, pois você precisa prestar atenção a cada nó em qual dimensão você está testando). Mas a pesquisa do vizinho mais próximo é mais complicada. À medida que descemos as ramificações, dividindo os exemplos ao meio, em alguns casos podemos descartar a outra metade dos exemplos. Mas nem sempre. Às vezes, o ponto que estamos consultando cai muito próximo do limite de divisão. O ponto de consulta em si pode ser no lado esquerdo do limite, mas um ou mais dos k vizinhos mais próximos podem realmente estar no lado direito. Temos que testar essa possibilidade calculando a distância do ponto de consulta até o limite de divisão, e depois buscar em ambos os lados se não pudermos encontrar os k exemplos à esquerda que estão mais perto do que essa distância. Devido a esse problema, as árvores k-d são apropriadas somente quando houver muito mais exemplos que dimensões, de preferência pelo menos 2^n exemplos. Assim, as árvores k-d funcionam bem com até 10 dimensões com milhares de exemplos ou até 20 dimensões com milhões de exemplos. Se não tivermos exemplos suficientes, a pesquisa não será mais rápida do que uma varredura linear do conjunto de dados inteiro.

18.8.3 Hash sensível a localidade

As tabelas hash têm o potencial de fornecer pesquisa ainda mais rápida do que as árvores binárias. Mas como podemos encontrar os vizinhos mais próximos usando uma tabela hash quando os códigos de hash dependem de uma correspondência exata? Os códigos de hash distribuem os valores aleatoriamente entre os compartimentos, mas queremos ter pontos próximos agrupados no mesmo compartimento; queremos um **hash sensível a localidade** (HSL).

Não podemos usar tabelas de hash para resolver $VP(k, \mathbf{x}_q)$ exatamente, mas com um uso inteligente de algoritmos randomizados podemos encontrar uma solução *aproximada*. Primeiro vamos definir o problema dos **vizinhos próximos aproximados**: dado um conjunto de dados de pontos de exemplo e um ponto de consulta $\mathbf{x}_q$, encontre, com probabilidade alta, um ponto de exemplo (ou pontos) que esteja próximo de $\mathbf{x}_q$. Para ser mais preciso, é necessário que, se houver um ponto $\mathbf{x}_j$, que esteja dentro de um raio de r de $\mathbf{x}_q$, é muito provável que na sequência, o algoritmo vai encontrar um ponto de $\mathbf{x}_j$’ que esteja dentro da distância de $c\ r$ de θ. Se não houver ponto dentro do raio r é permitido que o algoritmo relate falha. Os valores de c e “alta probabilidade” são os parâmetros do algoritmo.

Para resolver aproximar vizinhos próximos, vamos precisar de uma função hash $g(\mathbf{x})$ que tenha a propriedade de que, para quaisquer dois pontos $\mathbf{x}_j$ e $\mathbf{x}_{j'}$, a probabilidade de ter o mesmo código hash será pequena se a sua distância for superior a $c\ r$, e alta, se a distância for menor que r. Para simplificar vamos tratar cada ponto como uma cadeia de bits (quaisquer recursos que não forem booleanos podem ser codificados em um conjunto de características booleanas).

A intuição com a qual contamos é que, se dois pontos estão juntos em um espaço n-dimensional, então eles necessariamente vão estar perto quando projetados em um espaço unidimensional (uma linha). Na verdade, podemos discretizar a linha em compartimentos de hash de modo que, com alta probabilidade, os pontos próximos serão representados exatamente no mesmo compartimento. Os pontos que estiverem longe uns dos outros tendem a ser representados em compartimentos diferentes para a maioria das projeções, mas haverá sempre poucas representações que coincidentemente projetam os pontos separados no mesmo compartimento. Assim, o compartimento do ponto $\mathbf{x}_q$ contém muitos (mas não todos) os pontos que estão perto de $\mathbf{x}_q$, bem como alguns pontos que estão longe.

O truque de HSL é criar de *múltiplas* representações aleatórias e combiná-las. A representação aleatória é apenas um subconjunto aleatório de representação da cadeia de bits. Escolhemos l diferentes representações aleatórias e a criação de l tabelas hash, $g_1(\mathbf{x}),\ldots,\ g_l(\mathbf{x})$. Em seguida, inserimos todos os exemplos em cada tabela hash. Então, quando é dado um ponto de consulta $\mathbf{x}_p$, buscamos o conjunto de pontos no compartimento $g_k(\theta)$ para cada k e unimos esses conjuntos em um conjunto de pontos candidatos, C. Então calculamos a distância real até $\mathbf{x}_q$ para cada um dos pontos em C e retornamos os pontos k mais próximos. Com probabilidade alta, cada um dos pontos que estão perto de $\mathbf{x}_q$ vai aparecer em pelo menos um dos compartimentos e, apesar de alguns pontos distantes também aparecerem, podemos ignorá-los. Para problemas grandes do mundo real, tal como encontrar os vizinhos mais próximos em um conjunto de dados de 13 milhões de imagens da Web utilizando 512 dimensões (Torralba *et al.*, 2008), o HSL precisa examinar apenas alguns milhares de imagens de 13 milhões para encontrar os vizinhos mais próximos — um aumento de velocidade de mil vezes sobre as abordagens exaustivas ou de árvore k-d.

18.8.4 Regressão não paramétrica

Agora vamos examinar as abordagens não paramétricas para *regressão* em vez de classificação. A Figura 18.28 mostra um exemplo de alguns modelos diferentes. Em (a), temos talvez o método mais simples de todos, conhecido informalmente como "ligar os pontos" e, de forma esnobe, como "regressão não paramétrica linear programável por partes". Esse modelo cria uma função $h(x)$ que, dada uma consulta $\mathbf{x}_q$, resolve o problema de regressão linear simples com apenas dois pontos: exemplos de treinamento imediatamente à esquerda e à direita de $\mathbf{x}_q$. Quando o ruído for baixo, na verdade esse método corriqueiro não é tão ruim, razão pela qual é um recurso padrão de software de gráficos em planilhas. Mas, quando os dados são ruidosos, a função resultante é pontuda e não generaliza bem.

A **regressão de *k*-vizinhos mais próximos** (Figura 18.28(b)) melhora ao ligar os pontos. Em lugar

de usar apenas os dois exemplos para a esquerda e para a direita de um ponto de consulta $\mathbf{x}_q$, usaremos os k vizinhos mais próximos (aqui, k-3). Um valor maior de k tende a suavizar a magnitude das pontas, embora a função resultante tenha descontinuidades. Em (b), temos a média de k-vizinhos mais próximos: $h(x)$ é o valor médio dos pontos k, $\sum y_j/k$. Repare que, nos pontos de periferia, perto de $x = 0$ e $x = 14$, as estimativas são fracas porque toda evidência vem de um lado (o interior) e ignora a tendência. Em (c), temos a regressão linear de k-vizinhos mais próximos, que encontra a melhor linha através dos exemplos k. Este faz um trabalho melhor de tendência de captura dos que estão à parte, mas ainda é descontínuo. Em (b) e (c), ficamos com a questão de como escolher um bom valor para k. A resposta, como usual, é a validação cruzada.

A **regressão ponderada localmente** (Figura 18.28(d)) oferece-nos as vantagens do vizinho mais próximo, sem descontinuidades. Para evitar descontinuidades em $h(x)$, precisamos evitar descontinuidades no conjunto de exemplos que usamos para estimar $h(x)$. A ideia de regressão ponderada localmente é que, em cada ponto da consulta x_q, os exemplos que estão perto de x_q são fortemente ponderados, e os exemplos que estão mais longe são ponderados menos intensamente ou não são. A redução do peso sobre a distância é sempre gradual, não repentina.

Decidimos o quanto ponderar cada exemplo com uma função conhecida como **kernel**. A função kernel é parecida com um calombo; na Figura 18.29 vemos o kernel específico usado para gerar a Figura 18.28(d). Podemos ver que o peso fornecido por esse kernel é maior no centro e chega a zero a uma distância de ±5. Podemos escolher qualquer função para um kernel? Não. Primeiro, observe que chamamos uma função kernel $\mathcal{K}$ com $\mathcal{K}$ (*Distância*($\mathbf{x}_j$, $\mathbf{x}_q$)), onde $\mathbf{x}_q$ é um ponto de consulta que está a determinada distância de $\mathbf{x}_j$, e queremos saber o quanto ponderar essa distância. Então, $\mathcal{K}$ deve ser simétrico em torno de 0 e ter um máximo em 0. A área sob o kernel deve permanecer limitada à medida que avançamos para ±∞. Outras formas, tais como as gaussianas, têm sido utilizadas para kernels, mas as últimas pesquisas sugerem que a escolha da forma não importa muito. Temos de ter cuidado com a largura do kernel. Novamente, esse é um parâmetro do modelo que pode ser escolhido melhor por validação cruzada. Assim como na escolha do k dos vizinhos mais próximos, se os kernels forem demasiado grandes vamos chegar a uma subadaptação e, se eles forem demasiados estreitos, teremos uma superadaptação. Na Figura 18.29(d), o valor de $k = 10$ fornece uma curva suave que parece boa, mas talvez sem dar atenção suficiente para o ponto fora da curva em $x = 6$; uma largura menor do kernel seria mais sensível aos pontos individuais.

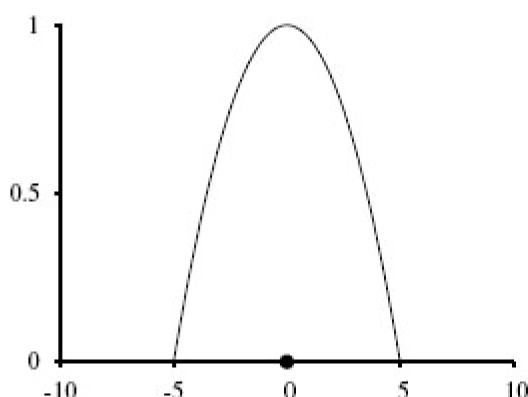


Figura 18.29 Um kernel quadrático, $\mathcal{K}(d) = \max(0,1 - (2\,|x|/k)^2)$, com largura do kernel com $k = 10$, centralizado no ponto de consulta $x = 0$.

Fazer regressão ponderada localmente com kernels fica simples agora. Para determinado ponto de consulta $\mathbf{x}_q$ vamos resolver o seguinte problema de regressão ponderada usando a descida pelo gradiente:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j \mathcal{K}(\,Distância\,(\mathbf{x}_q, \mathbf{x}_j))\,(y_j - \mathbf{w} \cdot \mathbf{x}_j)^2 \;,$$

onde a *Distância* é uma das métricas de distância discutidas para vizinhos mais próximos. Então, a resposta é $h(\mathbf{x}_q) = \mathbf{w}^* \cdot \mathbf{x}_q$.

Observe que precisamos resolver um problema de regressão para *cada* ponto de consulta — isso é o que significa ser local (na regressão linear simples, resolvemos o problema de regressão uma vez, globalmente, e depois usamos o mesmo $h_{\mathbf{w}}$ para qualquer ponto da consulta). Alivia esse trabalho extra o fato de que cada problema de regressão será mais fácil de resolver, pois envolve apenas os exemplos com o peso diferente de zero — exemplos cujos kernels se sobrepõem aos pontos de consulta atenuam esse trabalho extra. Quando a largura do kernel é pequena, podem ser apenas alguns pontos.

A maioria dos modelos não paramétricos tem a vantagem de tornar fácil fazer a validação cruzada com omissão de um sem ter de recalcular tudo. Com um modelo de k-vizinhos mais próximos, por exemplo, quando é dado um exemplo de teste $(\mathbf{x}, y)$ recuperamos os k vizinhos mais próximos uma vez, calculamos, por exemplo, a perda $L(y, h(\mathbf{x}))$ deles e registramos isso como o resultado de omissão de um para cada exemplo que não for um dos vizinhos. Então recuperamos os vizinhos mais próximos $k + 1$ e registramos os resultados distintos omitindo cada um dos k vizinhos. Com N exemplos, o processo todo é $O(k)$, não $O(kN)$.

18.9 MÁQUINAS DE VETORES DE SUPORTE

A **máquina de vetor de suporte** ou estrutura SVM é atualmente a abordagem pré-fabricada mais popular para aprendizagem supervisionada: se você não tiver nenhum conhecimento prévio especializado sobre um domínio, o SVM é um excelente primeiro método a testar. Existem três propriedades que tornam o SVM atraente:

1. Os SVMs constroem um **separador de margem máxima** — um limite de decisão com a maior distância possível a pontos de exemplo. Isso os ajuda a generalizar bem.
2. Os SVMs criam uma separação linear em hiperplano, mas têm a capacidade de incorporar os dados em um espaço de dimensão superior, usando o assim chamado **truque de kernel**. Muitas vezes, os dados que não são separáveis linearmente no espaço de entrada original são facilmente separáveis em um espaço de dimensão superior. O separador linear de dimensão superior é realmente não linear no espaço original. Isso significa que o espaço de hipótese é expandido em relação aos métodos que usam representações estritamente lineares.
3. Os SVMs são um método não paramétrico — eles mantêm exemplos de treinamento e podem precisar armazenar todos eles. Por outro lado, na prática, acabam mantendo apenas uma

pequena fração do número de exemplos — às vezes apenas uma constante pequena vezes o número de dimensões. Assim, os SVMs combinam as vantagens de modelos não paramétricos e paramétricos: eles têm a flexibilidade para representar funções complexas, mas são resistentes à superadaptação.

Pode-se dizer que os SVMs são bem-sucedidos devido a uma ideia básica e um truque hábil. Abrangeremos um de cada vez. Na Figura 18.30(a), temos um problema de classificação binária com três limiares de decisão candidatos, cada um deles um separador linear. Todos são compatíveis com todos os exemplos; por isso, do ponto de vista da perda 0/1, cada um seria igualmente bom. A regressão logística encontrará alguma linha de separação; a localização exata da linha depende de todos os pontos de exemplo. A ideia dos SVMs é que alguns exemplos são mais importantes que os outros e prestar atenção a eles pode levar a melhor generalização.

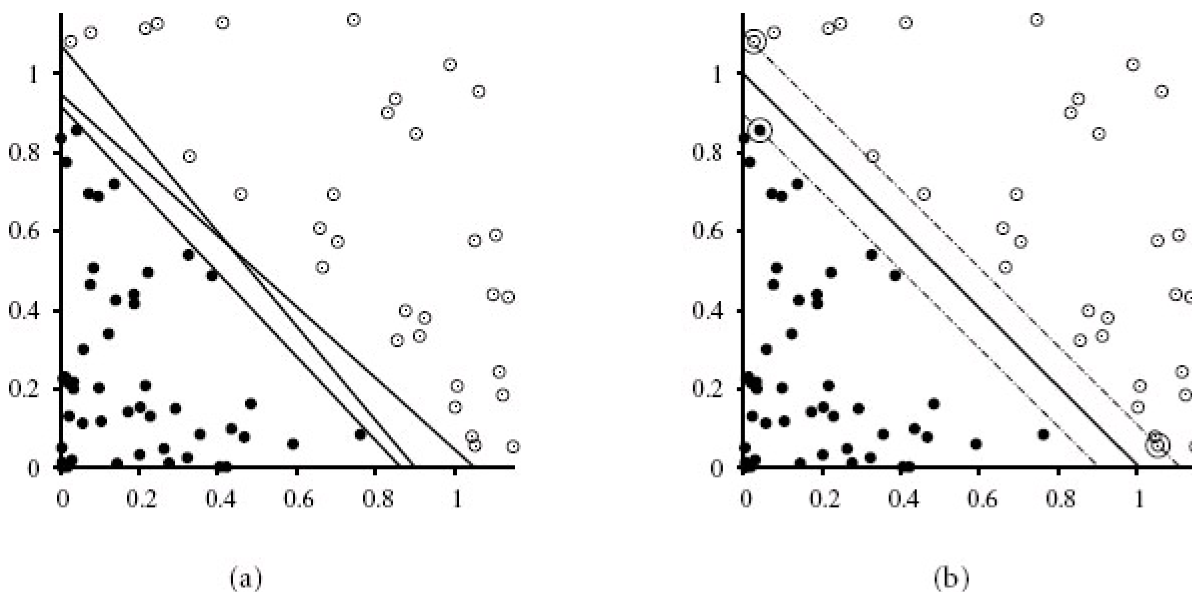


Figura 18.30 Máquina de vetor de suporte. (a) Duas classes de pontos (preto e círculos brancos) e três separadores lineares candidatos. (b) O separador de margem máxima (linha pesada) está no ponto médio da margem (área entre as linhas tracejadas). O **suporte vetorial** (pontos com grandes círculos) é o exemplo mais próximo do separador.

Considere a mais baixa de três linhas de separação em (a). Ela chega muito próximo a 5 dos exemplos pretos. Apesar de classificar todos os exemplos corretamente e, portanto, minimizar a perda, ele deve deixá-lo nervoso por tantos exemplos estarem perto da linha; pode ser que outros exemplos pretos fiquem do outro lado da linha.

Os SVMs abordam esta questão assim: em vez de minimizar a *perda empírica* esperada sobre dados de treinamento, eles tentam minimizar a perda de *generalização* esperada. Não sabemos onde podem cair os pontos ainda não vistos, mas, sob o pressuposto probabilístico de que eles são extraídos da mesma distribuição que os exemplos vistos anteriormente, existem alguns argumentos da teoria da aprendizagem computacional (Seção 18.5) sugerindo que minimizemos a perda de generalização escolhendo o separador que está mais distante dos exemplos que temos visto até agora. Chamamos esse separador, mostrado na Figura 18.30(b), de **separador de margem máxima**. A **margem** é a largura da zona delimitada pelas linhas tracejadas na figura — duas vezes a distância do separador até o ponto de exemplo mais próximo.

Agora, como encontraremos esse separador? Antes de mostrar as equações, alguma notação: tradicionalmente, os SVMs usam a convenção de que os rótulos de classe são +1 e −1, em vez de +1 e 0, que temos usado até agora. Além disso, embora coloquemos o corte no vetor de peso **w** (e um valor fictício 1 correspondente em $x_{j,0}$) os SVMs não fazem isso; eles mantêm o corte como um parâmetro separado b. Com isso em mente, o separador é definido como o conjunto de pontos $\{\mathbf{x}: \mathbf{w} \cdot \mathbf{x} + b = 0\}$. Poderíamos procurar o espaço de **w** e b com a descida pelo gradiente para encontrar os parâmetros que maximizam a margem e ao mesmo tempo classificar corretamente todos os exemplos.

No entanto, verifica-se que há outra abordagem para resolver esse problema. Não vamos mostrar os detalhes, apenas dizer que há uma representação alternativa chamada de representação dual, em que a solução ótima é encontrada resolvendo

$$\underset{\alpha}{\operatorname{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k) \tag{18.13}$$

sujeito às restrições $\alpha_j \geq 0$ e $\sum_j \alpha_j y_j = 0$. Esse é um problema de otimização de **programação quadrática**, para o qual existem bons pacotes de software. Uma vez encontrado o vetor, podemos voltar a **w** com a equação $\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j$ ou podemos ficar em representação dual. Existem três propriedades importantes da Equação 18.13. Em primeiro lugar, a expressão é convexa, tem um único máximo global que pode ser encontrado de forma eficiente. Segundo, os dados *inserem a expressão apenas na forma de produtos escalares de pares de pontos.* Essa segunda propriedade é também verdadeira para a equação do separador em si; uma vez que o a_j ótimo foi calculado, é

$$h(\mathbf{x}) = \text{sinal} \left(\sum_j \alpha_j y_j (\mathbf{x} \cdot \mathbf{x}_j) - b \right) . \tag{18.14}$$

Uma propriedade final importante é que os pesos j associados a cada ponto de dados são *zero*, exceto pelos **vetores de suporte** — os pontos mais próximos do separador (eles são chamados de vetores de "suporte" porque "sustentam" o plano de separação). Como, normalmente, há muito menos vetores de suporte que exemplos, os SVMs obtêm algumas das vantagens dos modelos paramétricos.

E se os exemplos não fossem linearmente separáveis? A Figura 18.31(a) mostra um espaço de entrada definido por atributos $\mathbf{x} = (x_1, x_2)$, com exemplos positivos ($y = +1$) dentro de uma região circular e exemplos negativos ($y = -1$) fora. Certamente não há separador linear para esse problema. Suponha agora que expressemos novamente os dados de entrada, ou seja, façamos o mapeamento de cada vetor de entrada **x** para um novo vetor de valores característicos, $F(\mathbf{x})$. Em particular, vamos usar as três características

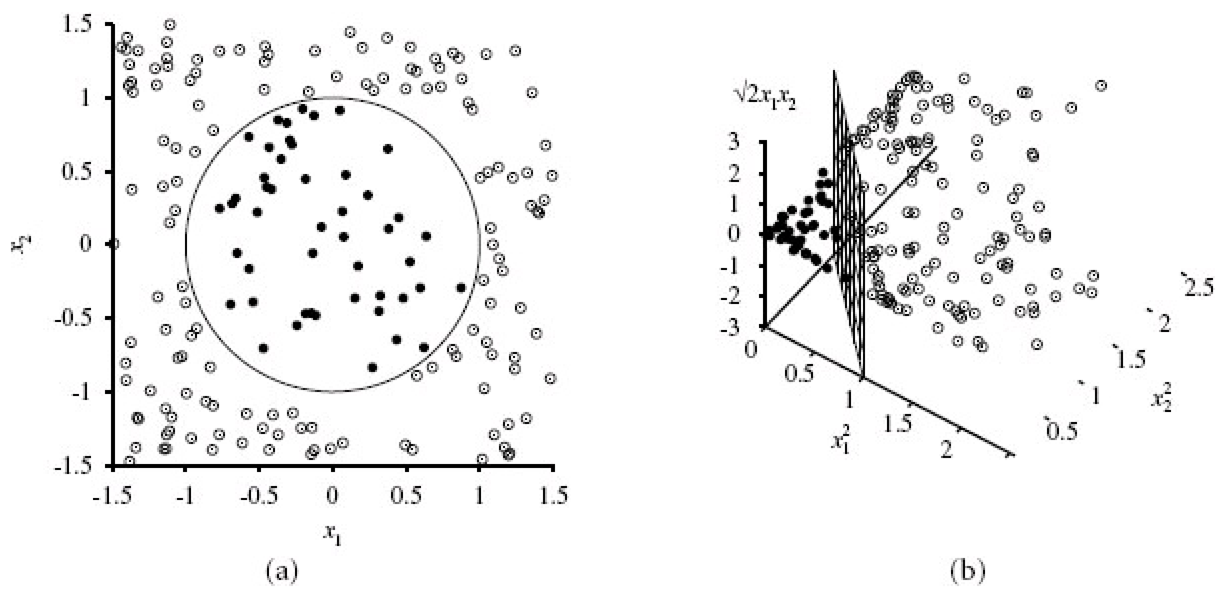


Figura 18.31 (a) Um conjunto de treinamento em duas dimensões com exemplos positivos como círculos pretos e exemplos negativos como círculos brancos. A fronteira de decisão verdadeira também é exibida, $x_1^2 + x_2^2 \leq 1$. (b) Os mesmos dados após o mapeamento em um espaço de entrada tridimensional $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. A fronteira de decisão circular em (a) torna-se uma fronteira de decisão linear em três dimensões. A Figura 18.30(b) dá um *close-up* do separador em (b).

$$f_1 = x_1^2 , \qquad f_2 = x_2^2 , \qquad f_3 = \sqrt{2}x_1x_2 . \tag{18.15}$$

Veremos em breve de onde eles vieram, mas por ora basta olhar o que acontece. A Figura 18.31(b) mostra os dados no espaço novo, tridimensional, definido pelas três características; os dados são *linearmente separáveis* nesse espaço! Esse fenômeno é realmente bastante geral: se os dados forem mapeados em um espaço de dimensão suficientemente alta, eles serão quase sempre linearmente separáveis — se você olhar para um conjunto de pontos de direções suficientes, encontrará uma maneira de fazê-los alinhar-se. Aqui, nós usamos apenas três dimensões;[11] o Exercício 18.16 pede para mostrar que quatro dimensões são suficientes para separar linearmente um círculo em qualquer lugar no plano (não apenas na origem) e que cinco dimensões são suficientes para separar linearmente qualquer elipse. Em geral (com alguns casos especiais de exceção), se tivermos N pontos de dados, eles serão sempre separados em espaços de $N - 1$ ou mais dimensões (Exercício 18.25).

Normalmente, não esperaríamos encontrar um separador linear no espaço de entrada $\mathbf{x}$, mas podemos encontrar separadores lineares no espaço $F(\mathbf{x})$ de dimensão superior simplesmente substituindo $\mathbf{x}_j \cdot \mathbf{x}_k$ na Equação 18.13 com $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$. Isso, por si só, não é notável — substituir $\mathbf{x}$ por $F(\mathbf{x})$ em qualquer algoritmo de aprendizagem tem o efeito requerido —, mas o produto escalar tem algumas propriedades especiais. Acontece que $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$ muitas vezes pode ser calculado sem antes calcular F para cada ponto. Em nosso espaço característico tridimensional definido pela Equação 18.15, um pouco de álgebra mostra que

$$F(\mathbf{x}_j) \cdot F(\mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2 .$$

(Essa é a razão pela qual $\sqrt{2}$ está em f_3.) A expressão $(\mathbf{x}_j \cdot \mathbf{x}_k)^2$ é chamada de **função kernel**,[12] e

geralmente é escrita como $K(\mathbf{x}_j, \mathbf{x}_k)$. A função kernel pode ser aplicada a pares de dados de entrada para avaliar produtos escalares em algum espaço característico correspondente. Assim, podemos encontrar separadores lineares em espaços característicos de dimensão superior $F(\mathbf{x})$ simplesmente substituindo $\mathbf{x}_j \cdot \mathbf{x}_k$ na Equação 18.13 com uma função kernel $K(\mathbf{x}_j, \mathbf{x}_k)$. Assim, podemos aprender no espaço de dimensão superior, mas calculamos apenas as funções kernel em vez de uma lista completa de características para cada ponto de dados.

A próxima etapa é verificar que não há nada especial sobre o kernel $K(\mathbf{x}_j, \mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2$. Corresponde a determinado espaço característico de dimensão superior, mas outras funções kernel correspondem a outros espaços característicos. Um resultado respeitável em matemática, o teorema de Mercer (1909) informa-nos que qualquer função kernel "razoável"[13] corresponde a *algum* espaço característico. Esses espaços característicos podem ser muito grandes, mesmo para kernels que parecem inócuos. Por exemplo, o **kernel polinomial** $K(\mathbf{x}_j, \mathbf{x}_k) = (1 + \mathbf{x}_j \cdot \mathbf{x}_k)^d$ corresponde a um espaço característico cuja dimensão é exponencial em d.

Esse é, então, o **truque de kernel** inteligente: ligando esses kernels na Equação 18.13 *podem ser encontrados ótimos separadores lineares de forma eficiente em espaços característicos com bilhões de dimensões (ou, em alguns casos, infinitamente muitas).* Os separadores lineares resultantes, quando mapeados de volta ao espaço de entrada original, podem corresponder a fronteiras de decisão arbitrariamente não lineares, sinuosas entre os exemplos positivos e negativos.

O caso de dados inerentemente ruidosos pode não querer um separador linear em algum espaço de dimensão superior. Em vez disso, gostaríamos de uma superfície de decisão em um espaço de dimensão inferior que não separe as classes claramente, mas reflita a realidade dos dados ruidosos. Isso é possível com o classificador de **margem suave**, que permite que os exemplos caiam no lado errado da fronteira de decisão, mas lhes atribui a penalidade proporcional à distância necessária para movê-los de volta ao lado correto.

O método de kernel pode ser aplicado não apenas a algoritmos de aprendizagem que encontram os melhores separadores lineares, mas também a qualquer outro algoritmo que possa ser reformulado para funcionar somente com produtos escalares de pares de pontos de dados, como nas Equações 18.13 e 18.14. Uma vez que isso seja feito, o produto escalar é substituído por uma função kernel e temos uma versão kernelizada do algoritmo. Isso pode ser feito facilmente para k-vizinhos mais próximos e para aprendizagem do perceptron (Seção 18.7.2), entre outros.

18.10 APRENDIZAGEM POR AGRUPAMENTO

Até agora, examinamos métodos para a aprendizagem em que uma única hipótese, escolhida a partir de um espaço de hipóteses, é usada para fazer previsões. A ideia de métodos de **aprendizagem por agrupamento** é selecionar uma coleção inteira ou um agrupamento de hipóteses, a partir do espaço de hipóteses, e combinar suas previsões. Por exemplo, durante a validação cruzada poderíamos gerar 20 árvores de decisão diferentes do mesmo conjunto de treinamento e depois fazê-las votar na melhor classificação para um novo exemplo.

A motivação para a aprendizagem por agrupamento é simples. Considere um conjunto de $K = 5$ hipóteses e suponha que combinamos suas previsões usando a votação por maioria simples. Para o conjunto classificar de forma incorreta um novo exemplo, *pelo menos três das cinco hipóteses têm de classificar o exemplo de modo incorreto*. A expectativa é que isso seja muito menos provável que uma classificação incorreta por uma única hipótese. Imagine a seguinte suposição: cada hipótese h_k no conjunto tem um erro p, isto é, a probabilidade de um exemplo escolhido ao acaso ser classificado de forma incorreta por h_k é p. Além disso, imagine que os erros cometidos por cada hipótese sejam *independentes*. Nesse caso, se p é pequeno, a probabilidade de ocorrer grande número de classificações incorretas é minúscula. Por exemplo, um simples cálculo (Exercício 18.18) mostra que usar um conjunto de cinco hipóteses reduz uma taxa de erros de 1 em10 a uma taxa de erros menor que 1 em 100. Agora, é óbvio que a suposição de independência é pouco razoável porque as hipóteses provavelmente serão iludidas do mesmo modo por quaisquer aspectos enganosos dos dados de treinamento. Porém, se as hipóteses forem pelo menos um pouco diferentes, reduzindo assim a correlação entre seus erros, a aprendizagem por agrupamento poderá ser muito útil.

Outro modo de pensar na ideia de conjunto é considerá-lo uma forma genérica de ampliar o espaço de hipóteses. Isto é, pense no próprio conjunto como uma hipótese e no novo espaço de hipóteses como o conjunto de todos os conjuntos possíveis que podem ser construídos a partir de hipóteses no espaço original. A Figura 18.32 mostra como isso pode resultar em um espaço de hipóteses mais expressivo. Se o espaço de hipóteses original permitir um algoritmo de aprendizagem simples e eficiente, o método de agrupamento fornecerá um caminho para se aprender uma classe muito mais expressiva de hipóteses, sem incorrer em muita complexidade computacional ou algorítmica adicional.

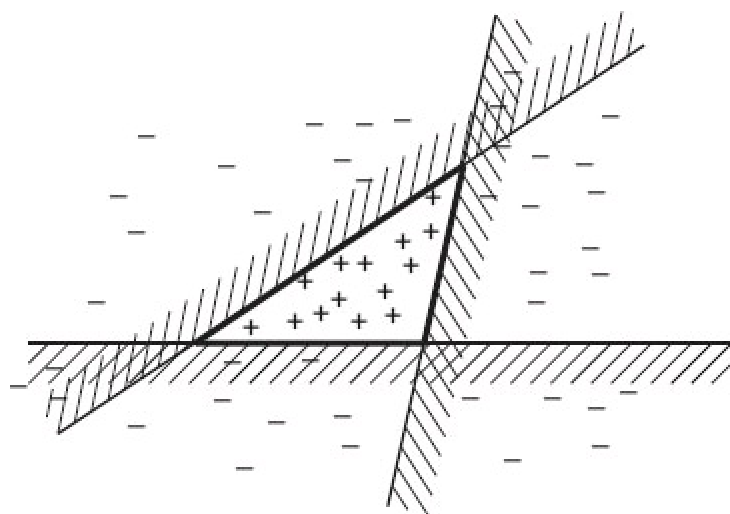


Figura 18.32 Ilustração do maior poder expressivo obtido pela aprendizagem por agrupamento. Adotamos três hipóteses de limiar linear, cada uma das quais classifica cada exemplo positivamente no lado não hachurado, e classificamos como positivo qualquer exemplo classificado positivamente pelas três hipóteses. A região triangular resultante é uma hipótese que não pode ser expressa no espaço de hipóteses original.

O método de agrupamento mais amplamente utilizado é chamado **aceleração**. Para entender como ele funciona, primeiro precisamos explicar a ideia de **conjunto de treinamento ponderado**. Em tal conjunto de treinamento, cada exemplo tem um peso associado $w_j \geq 0$. Quanto mais alto o peso de um exemplo, mais alta será a importância associada a ele durante a aprendizagem de uma hipótese. É simples modificar os algoritmos de aprendizagem que vimos até agora para operar com conjuntos de treinamento ponderados.[14]

A aceleração começa com $w_j = 1$ para todos os exemplos (isto é, um conjunto de treinamento normal). A partir desse conjunto, ele gera a primeira hipótese, h_1. Essa hipótese classificará alguns dos exemplos de treinamento de forma correta e outros de forma incorreta. Gostaríamos que a próxima hipótese classificasse melhor os exemplos incorretamente classificados e, assim, aumentamos seus pesos enquanto diminuímos os pesos dos exemplos corretamente classificados. A partir desse novo conjunto de treinamento ponderado, geramos a hipótese h_2. O processo continua desse modo até gerarmos K hipóteses, onde K é uma entrada para o algoritmo de aceleração. A hipótese de conjunto final é uma combinação de maioria ponderada de todas as K hipóteses, cada uma ponderada de acordo com o seu comportamento no conjunto de treinamento. A Figura 18.33 mostra como o algoritmo funciona conceitualmente. Existem muitas variantes da ideia básica de aceleração com diferentes modos de ajuste dos pesos e de combinação das hipóteses. Um algoritmo específico, denominado ADABOOST, é mostrado na Figura 18.34. Embora os detalhes dos ajustes de peso não sejam tão importantes, o ADABOOST tem uma propriedade muito importante: se o algoritmo de entrada de aprendizagem L é um algoritmo de **aprendizagem fraca** — o que significa que L sempre retorna uma hipótese com erro ponderado sobre o conjunto de treinamento que é ligeiramente melhor que o palpite aleatório (ou seja, 50% para classificação booleana) —, o ADABOOST retornará uma hipótese que classifica perfeitamente os dados de treinamento para K grande o bastante. Desse modo, o algoritmo acelera a exatidão do algoritmo de aprendizagem original sobre os dados de treinamento. Esse resultado é válido, independentemente de quanto o espaço de hipóteses original seja inexpressivo e de quanto seja complexa a função que está sendo aprendida.

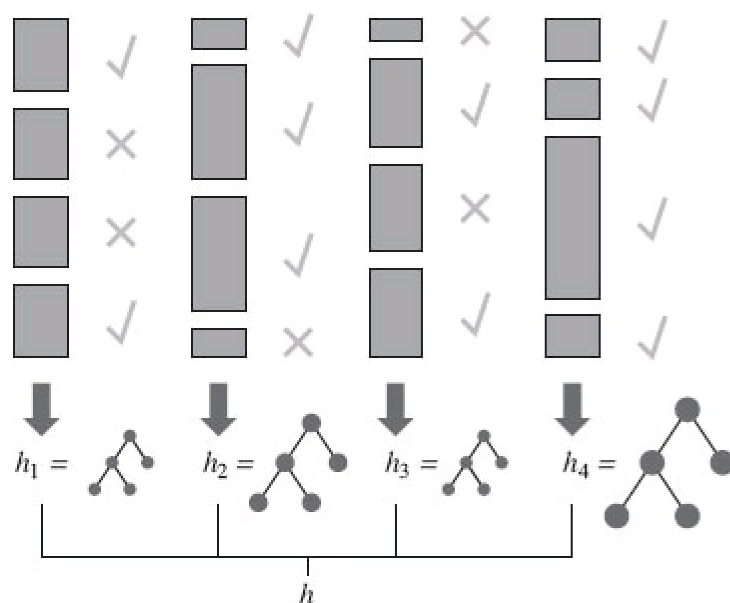


Figura 18.33 Como funciona o algoritmo de aceleração. Cada retângulo sombreado corresponde a um exemplo; a altura do retângulo corresponde ao peso. Os sinais de visto e os sinais cruzados indicam se o exemplo foi ou não classificado corretamente pela hipótese corrente. O tamanho da árvore de decisão indica o peso dessa hipótese no conjunto final.

```
função ADABOOST(exemplos, L,K) retorna uma hipótese de maioria ponderada
  entradas: exemplos, conjunto de N exemplos identificados (x1, y1), ..., (xN, yN)
            L, um algoritmo de aprendizagem
            K, o número de hipóteses no conjunto
  variáveis locais: w, um vetor de N pesos de exemplo, inicialmente 1/N
                    h, um vetor de K hipóteses
                    z, um vetor de K pesos de hipóteses
  para k = 1 até K faça
    h[k] ← L(exemplos, w)
    erro ← 0
    para j = 1 até N faça
      se h[k](xj) ≠ yj então erro ← erro + w[j]
    para j = 1 até N faça
    se h[k](xj) = yj então w[j] ← w[j] · erro/(1 – erro)
    w ← NORMALIZAR(w)
    z[k] ← log (1 – erro)/erro
  retornar MAIORIA-PONDERADA(h, z)
```

Figura 18.34 Variante ADABOOST do método de aceleração para aprendizagem por agrupamentos. O algoritmo gera hipóteses ao responder os exemplos um a um. A função MAIORIA-PONDERADA gera uma hipótese que retorna o valor de saída como voto mais alto entre as hipóteses em **h**, com os votos ponderados por **z**.

Vejamos como a aceleração se comporta sobre os dados de restaurante. Escolheremos como nosso

espaço de hipóteses original a classe de **cepos de decisão**, que são árvores de decisão com apenas um teste na raiz. A curva mais baixa na Figura 18.35(a) mostra que os cepos de decisão sem aceleração não são muito efetivos para esse conjunto de dados, alcançando desempenho de previsão de apenas 81% em 100 exemplos de treinamento. Quando a aceleração é aplicada (com K = 5), o desempenho é melhor, alcançando 93% depois de 100 exemplos.

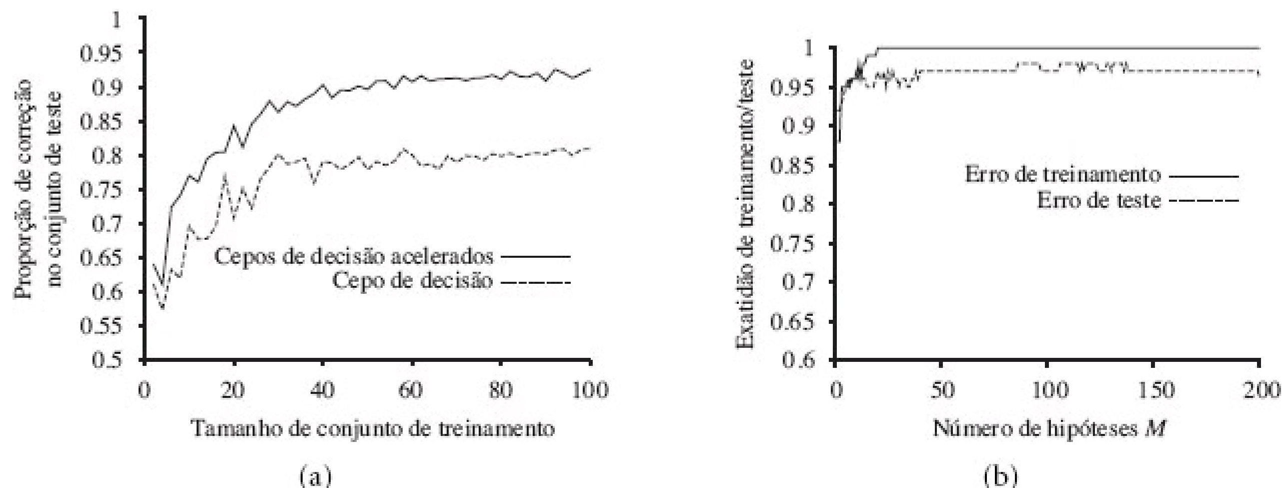


Figura 18.35 (a) Gráfico mostrando o desempenho de cepos de decisão acelerados com M = 5 *versus* cepos de decisão sobre os dados de restaurante. (b) A proporção correta no conjunto de treinamento, o conjunto de teste como função de M, o número de hipóteses no conjunto. Note que a exatidão do conjunto de teste melhora ligeiramente, mesmo após a exatidão do treinamento alcançar 1, isto é, depois que o conjunto se ajusta exatamente aos dados.

Um fato interessante ocorre à medida que o tamanho K do conjunto aumenta. A Figura 18.35(b) mostra o desempenho do conjunto de treinamento (em 100 exemplos) como uma função de K. Note que o erro alcança zero quando K é igual a 20, isto é, uma combinação ponderada pela maioria de 20 cepos de decisão basta para ajustar exatamente os 100 exemplos. À medida que mais cepos são adicionados ao conjunto, o erro permanece igual a zero. O grafo também mostra *que o desempenho do conjunto de teste continua a aumentar muito tempo depois de o erro do conjunto de treinamento ter alcançado zero*. Em K = 20, o desempenho do teste é 0,95 (ou 0,05 de erro) e o desempenho aumenta até 0,98 apenas quando K= 137, antes de cair gradualmente para 0,95.

Essa descoberta, bastante robusta entre conjuntos de dados e espaços de hipótese, surgiu como grande surpresa quando foi notada pela primeira vez. A navalha de Ockham nos diz que não devemos tornar as hipóteses mais complexas do que o necessário, mas o gráfico nos diz que as previsões melhoram à medida que a hipótese de conjunto fica mais complexa! Várias explicações foram propostas para isso. Uma visão é que a aceleração se aproxima da aprendizagem bayesiana (veja o Capítulo 20), que podemos mostrar ser um algoritmo de aprendizagem ótimo, e a aproximação melhora à medida que mais hipóteses são adicionadas. Outra explicação possível é que a inclusão de hipóteses adicionais permite ao conjunto ser mais definido em sua distinção entre exemplos positivos e negativos, o que ajuda quando se trata de classificar novos exemplos.

18.10.1 Aprendizagem on-line

Até agora, tudo o que fizemos neste capítulo baseou-se no pressuposto de que os dados são i.i.d. (independentes e identicamente distribuídos). Por um lado, isso é uma suposição sensata: se o futuro não tem nenhuma semelhança com o passado, como podemos prever alguma coisa? Por outro lado, há uma suposição muito forte: é raro que as nossas entradas tenham capturado todas as informações que tornariam o futuro verdadeiramente independente do passado.

Nesta seção, vamos examinar o que fazer quando os dados não são i.i.d., quando eles podem mudar ao longo do tempo. Nesse caso, importa *quando* faremos uma previsão, por isso vamos adotar a perspectiva chamada de **aprendizagem on-line**: um agente recebe uma entrada x_j da natureza, prevê o y_j correspondente e, em seguida, recebe a resposta correta. O processo então se repete com x_{j+1}, e assim por diante. Pode-se pensar que essa tarefa é impossível — se a natureza for contraditória, todas as previsões podem estar erradas. Acontece que podemos ter algumas garantias.

Vamos considerar a situação em que a nossa entrada consiste em previsões de um painel de especialistas. Por exemplo, a cada dia um conjunto de K especialistas prevê se o mercado de ações vai subir ou baixar, e nossa tarefa é estabelecer associação sobre essas previsões e torná-las nossas próprias. Uma maneira de fazer isso é acompanhar a *performance* de cada especialista e escolher confiar neles, na proporção de seus resultados passados. Isso é chamado de **algoritmo aleatório de maioria ponderada**. Podemos descrevê-los mais formalmente:

1. Inicializar com 1 um conjunto de pesos $\{w_1,\ldots, w_k\}$ todos com 1.
2. Receber as previsões $\{\hat{y}_1,\ldots,\hat{y}_K\}$ dos especialistas.
3. Escolher aleatoriamente um especialista k^*, em proporção ao seu peso: $P(k) = w_k/(\sum_{k'} w_{k'})$.
4. Prever $\hat{y}_{k^*}$.
5. Receber a resposta correta y.
6. Para cada especialista k tal que $\hat{y}_k \neq y$, atualizar $w_k \leftarrow \beta w_k$.

Aqui β é um número, de $0 < \beta < 1$, que informa o quanto penalizar um especialista por cada erro.

Medimos o sucesso desse algoritmo em termos de **arrependimento**, que é definido como o número de erros adicionais que fazemos em relação ao especialista que, em retrospectiva, teve o melhor registro de previsão. Seja M^* o número de erros cometidos pelos melhores especialistas. A seguir, o número de erros, M, cometidos pelo algoritmo aleatório de maioria ponderada, é limitado por[15]

$$M < \frac{M^* \ln(1/\beta) + \ln K}{1-\beta}.$$

Esse limite vale para *qualquer* sequência de exemplos, até mesmo os escolhidos pelos adversários tentando fazer o pior. Para ser mais específico, quando houver $K = 10$ especialistas, se escolhermos $\beta = 1/2$ o número de erros será delimitado por $1{,}39M^* + 4{,}6$ e, se $\beta = 3/4$ por $1{,}15M^* + 9{,}2$. Em geral, se β estiver próximo de 1, somos suscetíveis às mudanças em longo prazo; se o melhor especialista mudar, vamos assimilar antes que seja tarde. No entanto, pagamos uma penalidade no início, quando começamos igualmente com todos os especialistas de confiança; podíamos aceitar o conselho dos maus especialistas por muito tempo. Quando b está próximo de 0, esses dois fatores estão invertidos. Note que podemos escolher b para chegar assintoticamente perto de M^* em longo

prazo; isso se chama **aprendizagem sem nenhum arrependimento** (porque a quantidade média de arrependimento por tentativa tende a 0 à medida que o número de tentativas aumenta).

A aprendizagem on-line é útil quando os dados podem estar mudando rapidamente ao longo do tempo. Também é útil para aplicações que envolvem grande coleção de dados que está em constante crescimento, mesmo se as mudanças forem graduais. Por exemplo, com um banco de dados de milhões de imagens da Web, você não vai querer experimentar, digamos, um modelo de regressão linear em todos os dados e em seguida treinar de novo a partir do zero toda vez que uma imagem nova for adicionada. Seria mais prático ter um algoritmo on-line que permitisse que as imagens fossem acrescentadas aos poucos. Para a maioria dos algoritmos de aprendizagem baseados na minimização de perdas, existe uma versão on-line baseada na minimização do arrependimento. É um bônus que muitos desses algoritmos on-line vêm com limites garantidos contra o arrependimento.

Para alguns observadores, é surpreendente existir limites tão justos de quão bem se pode comparar a um quadro de especialistas. Para outros, é realmente surpreendente que, quando um quadro de especialistas humanos se reúne — previsão do preço das ações no mercado, resultados desportivos ou disputas políticas —, o público que observa é tão desejoso de vê-los acertar e de não quantificar as suas taxas de erro.

18.11 APRENDIZAGEM DE MÁQUINA NA PRÁTICA

Introduzimos ampla gama de técnicas de aprendizagem de máquina, cada uma ilustrada com tarefas simples de aprendizagem. Nesta seção, consideraremos dois aspectos da aprendizagem de máquina na prática. O primeiro envolve encontrar algoritmos capazes de aprender a reconhecer dígitos escritos à mão e tirar deles até a última gota de previsão de desempenho. O segundo envolve nada além de apontar que a obtenção, a limpeza e a representação dos dados é pelo menos tão importante quanto a engenharia do algoritmo.

18.11.1 Estudo de caso: o reconhecimento de dígitos escritos à mão

Reconhecer dígitos escritos à mão é um problema importante com muitas aplicações, incluindo classificação automática de correspondências por código postal, leitura automática de cheques e devolução de impostos, e entrada de dados para computadores manuais. É uma área que experimentou progresso rápido, em parte devido a melhores algoritmos de aprendizagem e em parte por causa da disponibilidade de melhores conjuntos de treinamento. O United States National Institute of Science and Technology (**NIST**) colocou em arquivo um banco de dados de 60.000 dígitos rotulados, cada um com $20 \times 20 = 400$ pixels com valores de 8 bits em tons de escala de cinza. Tornou-se um dos problemas de referência-padrão para a comparação de novos algoritmos de aprendizagem. Como exemplo são mostrados alguns dígitos na Figura 18.36.



Figura 18.36 Exemplos do banco de dados NIST de dígitos escritos à mão. Linha superior: exemplos de dígitos 0-9, fáceis de identificar. Linha inferior: exemplos dos mesmos dígitos, mais difíceis de identificar.

Foram tentadas muitas abordagens de aprendizagem diferentes. Uma das primeiras, e provavelmente a mais simples, é o classificador de **três vizinhos mais próximos**, que também tem a vantagem de não necessitar de tempo de treinamento. Como um algoritmo baseado em memória, no entanto, deve armazenar todas as 60.000 imagens, e seu desempenho em tempo de execução fica lento. Alcançou taxa de erro de teste de 2,4%.

Para esse problema foi projetada uma **rede neural de camada oculta única** com 400 unidades de entrada (uma por pixel) e 10 unidades de saída (uma por classe). Usando validação cruzada, verificou-se que cerca de 300 unidades ocultas apresentaram o melhor desempenho. Com interconexão total entre as camadas, havia um total de 123.300 pesos. Essa rede alcançou taxa de erro de 1,6%.

Uma série de **redes neurais especializadas** chamada LeNet foi concebida para tirar partido da estrutura do problema — que a entrada consistisse em pixels em uma matriz bidimensional e que pequenas mudanças na posição ou inclinação de uma imagem não fossem importantes. Cada rede tinha uma camada de entrada de 32 × 32 unidades, onde 20 × 20 pixels eram centrados de modo que cada unidade de entrada era apresentada com uma vizinhança local de pixels. Isso foi seguido por três camadas de unidades ocultas. Cada camada consistia em vários planos de matrizes $n \times n$, onde n era menor que a camada anterior para que a rede reduzisse a qualidade da amostragem de entrada e onde os pesos de cada unidade em um plano eram obrigados a ser idênticos, de modo que o plano agisse como um detector de característica: podia escolher uma característica, como uma longa linha vertical ou um arco semicircular curto. A camada de saída tinha 10 unidades. Muitas versões dessa arquitetura foram experimentadas, uma representativa tinha camadas ocultas com 768, 192 e 30 unidades, respectivamente. O conjunto de treinamento foi aumentado através da aplicação de transformações afins para as entradas reais: deslocamento, rotação suave e dimensionamento de imagens (certamente as transformações tinham de ser pequenas ou um 6 era transformado em um 9!). A melhor taxa de erro alcançado pela LeNet foi de 0,9%.

Uma **rede neural acelerada** combinou três cópias da arquitetura LeNet, com uma segunda treinada em uma mistura de padrões que a primeira obtendo 50% de erro, e a terceira treinada em padrões dos quais as duas primeiras discordavam. Durante o teste, as três redes votaram com a decisão da maioria. A taxa de erro de teste foi de 0,7%.

Uma **máquina de vetor de suporte** (veja a Seção 18.9) com 25.000 vetores de suporte alcançou taxa de erro de 1,1%. Isso é notável porque a técnica SVM, como a abordagem simples de vizinho mais próximo que quase não exige raciocínio ou experimentação iterada por parte do desenvolvedor, ainda chegou perto do desempenho de LeNet, que teve anos de desenvolvimento. De fato, a máquina

de vetor de suporte não faz uso da estrutura do problema, e terá um bom desempenho se os pixels forem apresentados em ordem permutada.

Uma **máquina de vetor de suporte virtual** inicia com um SVM regular e, em seguida, o melhora com uma técnica projetada para tirar vantagem da estrutura do problema. Em vez de permitir produtos de todos os pares de pixels, essa abordagem concentra-se em kernels formados a partir de pares de pixels próximos. Também aumenta o conjunto de treinamento com as transformações dos exemplos, como o LeNet fazia. A SVM virtual alcançou a melhor taxa de erro registrada até hoje, 0,56%.

O **casamento de formas** é uma técnica de visão computacional usada para alinhar as partes correspondentes de duas imagens diferentes de objetos (Belongie *et al*., 2002). A ideia é escolher um conjunto de pontos em cada uma das duas imagens, e depois calcular, para cada ponto na primeira imagem, a qual ponto da segunda imagem corresponde. A partir desse alinhamento, calculamos uma transformação entre as imagens. A transformação nos dá uma medida da distância entre as imagens. Essa medida de distância é mais motivadora do que apenas contar o número de pixels diferentes, e verifica-se que o algoritmo 3-vizinho mais próximo, usando essa medida de distância, executa muito bem. Em um treinamento em apenas 20.000 dos 60.000 dígitos, e usando 100 pontos de amostra por imagem extraída de um detector de bordas Canny, um classificador de encaixe de formas alcançou erro de teste de 0,63%.

Estima-se que os **seres humanos** tenham taxa de erro de cerca de 0,2% sobre esse problema. Essa quantidade é um pouco suspeita porque os seres humanos não foram testados de forma tão extensiva quanto os algoritmos de aprendizagem de máquina. Em um conjunto de dados de dígitos semelhantes do correio americano, os erros humanos foram de 2,5%.

A tabela a seguir resume as taxas de erro, desempenho de tempo de execução, requisitos de memória e a quantidade de tempo de treinamento para os sete algoritmos que discutimos. Adiciona também outra medida: o percentual de dígitos que deve ser rejeitado para alcançar 0,5% de erro. Por exemplo, se for permitido que o SVM rejeite 1,8% das entradas, ou seja, passe-as para que alguém faça o julgamento final, sua taxa de erro em 98,2% do restante da entrada será reduzida de 1,1% para 0,5%.

A tabela a seguir resume a taxa de erro e algumas das outras características das sete técnicas que discutimos.

	3 *RN*	300 Oculto	LeNet	LeNet Acelerado	SVM	SVM Virtual	Casamento de Formas
Taxa de erro (pct.)	2,4	1,6	0,9	0,7	1,1	0,56	0,63
Tempo de execução (milissegundos/dígito)	1.000	10	30	50	2000	200	
Requisitos de memória (Mbyte)	12	0,49	0,012	0,21	11		
Tempo de treinamento (dias)	0	7	14	30	10		
% rejeitados para atingir 0,5% de erro	8,1	3,2	1,8	0,5	1,8		

18.11.2 Estudo de caso: sentido das palavras e preços das casas

Para simular a ideia de um livro, vamos lidar com dados simples: um conjunto pequeno de dados, geralmente em duas dimensões. Em aplicações práticas de aprendizagem de máquina, entretanto, o conjunto de dados geralmente é grande, multidimensional e bagunçado. Os dados não são entregues ao analista em um conjunto pré-empacotado de valores (**x**, *y*), mas o analista terá de conseguir os dados corretos. Há uma tarefa a ser cumprida, e a maioria dos problemas de engenharia é decidir que dados são necessários para realizar a tarefa; a menor parte é a escolha e a implementação de um método de aprendizagem de máquina apropriado para processar os dados. A Figura 18.37 mostra um exemplo típico do mundo real, comparando cinco algoritmos de aprendizagem na tarefa da classificação do sentido da palavra (dada uma sentença como "O banco dobrou", classificar a palavra "banco" como "moeda bancária" ou "banco de rio"). O ponto é que os pesquisadores de aprendizado de máquina têm focado principalmente na direção vertical: posso inventar um novo algoritmo de aprendizagem que desempenha melhor do que os algoritmos publicados anteriormente em um conjunto de treinamento padrão de um milhão de palavras? Mas o grafo mostra que há mais espaço para melhorias na direção horizontal: em vez de inventar um novo algoritmo, tudo o que é necessário é juntar 10 milhões de palavras de dados de treinamento; mesmo o *pior* algoritmo com 10 milhões de palavras desempenha melhor do que o melhor algoritmo com um milhão. Enquanto reunimos ainda mais dados, as curvas continuam a subir, superando as diferenças entre os algoritmos.

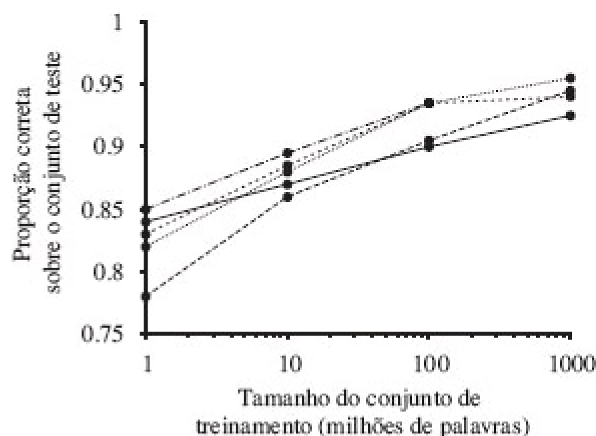


Figura 18.37 Curvas de aprendizagem de cinco algoritmos de aprendizagem em uma tarefa comum. Observe que parece haver mais espaço para melhoria na direção horizontal (mais dados de treinamento) do que na direção vertical (algoritmo de aprendizagem de máquina diferente). Adaptada de Banko e Brill (2001).

Considere outro problema: a tarefa de estimar o valor verdadeiro das casas que estão à venda. Na Figura 18.13 mostramos uma versão em miniatura do problema, fazendo a regressão linear do tamanho da casa pelo preço de compra. Você deve ter notado muitas limitações desse modelo. Primeiro, está medindo a coisa errada: queremos estimar o preço de venda de uma casa, não o preço de compra. Para resolver essa tarefa vamos precisar de dados sobre as vendas reais. Mas isso não significa que devemos desconsiderar os dados de preço de compra — podemos usá-los como um dos recursos de entrada. Além do tamanho da casa, vamos precisar de mais informações: o número de salas, quartos e banheiros, se a cozinha e os banheiros foram reformados recentemente, a idade da casa; também vamos precisar de informações sobre o lote e a vizinhança. Mas como definir vizinhança? Pelo CEP? E se parte do CEP estiver do lado "errado" da estrada ou do trilho do tem, e

a outra parte é que é desejável? E sobre o distrito escolar? O *nome* do distrito escolar deve ser uma característica ou os *resultados médios dos testes*? Além de decidir que características incluir, teremos de lidar com dados faltantes; áreas diferentes têm costumes diferentes sobre que dados devem ser relatados e, em casos individuais, estará sempre faltando alguns dados. Se os dados que deseja não estão disponíveis, talvez você possa estabelecer um site de rede social para incentivar as pessoas a compartilharem e corrigirem os dados. No final, esse processo de decidir que características usar, e como usá-las, é tão importante como escolher entre regressão linear, árvores de decisão ou alguma outra forma de aprendizagem.

Dito isto, deve-se escolher um método (ou métodos) para um problema. Não há garantia de escolher o melhor método, mas há algumas diretrizes rudimentares. As árvores de decisão são ideais quando há uma porção de características discretas e você acredita que muitas delas possam ser irrelevantes. Os métodos não paramétricos são bons quando há uma porção de dados e nenhum conhecimento anterior, e quando não se quer se preocupar muito com a escolha das características corretas (desde que haja menos de 20). Entretanto, os métodos não paramétricos, geralmente, fornecem uma função h que é mais cara para ser executada. As máquinas de vetores de suporte são muitas vezes consideradas o melhor método para a primeira tentativa, desde que o conjunto de dados não seja muito grande.

18.12 RESUMO

Este capítulo se concentrou na aprendizagem indutiva de funções determinísticas a partir de exemplos. Os principais pontos foram:

- A aprendizagem assume muitas formas, dependendo da natureza do agente, do componente a ser aperfeiçoado e da realimentação disponível.
- Se a realimentação disponível fornece a resposta correta para o exemplo de entrada, o problema de aprendizagem será chamado **aprendizagem supervisionada**. A tarefa é aprender uma função $y = h(x)$. A aprendizagem de uma função de valores discretos é chamada de **classificação**; a aprendizagem de uma função contínua é chamada de **regressão**.
- A aprendizagem indutiva envolve encontrar uma hipótese consistente que concorde com os exemplos. A **navalha de Ockham** sugere escolher a hipótese consistente mais simples. A dificuldade dessa tarefa depende da representação escolhida.
- As **árvores de decisão** podem representar todas as funções booleanas. A heurística de **ganho de informações** fornece um método eficiente para encontrar uma árvore de decisão simples e consistente.
- O desempenho de um algoritmo de aprendizagem é medido pela **curva de aprendizagem**, que mostra a exatidão da previsão no **conjunto de teste** como uma função do tamanho do **conjunto de treinamento**.
- Quando existem vários modelos para escolher, pode-se utilizar a **validação cruzada** para selecionar um modelo que vai generalizar bem.
- Às vezes, nem todos os erros são iguais. A **função de perda** informa o quão ruim é cada erro; o objetivo é, então, minimizar a perda de um conjunto de validação.

- A **teoria de aprendizagem computacional** analisa a complexidade de amostra e a complexidade computacional da aprendizagem indutiva. Existe um compromisso entre a expressividade da linguagem de hipóteses e a facilidade de aprendizagem.
- A **regressão linear** é um modelo amplamente utilizado. Os parâmetros ótimos de um modelo de regressão linear podem ser encontrados pela busca do gradiente de descida ou calculados com exatidão.
- Um classificador linear com um limiar difícil — também conhecido como **perceptron** — pode ser treinado por uma simples regra de atualização de peso para ajustar os dados que são **linearmente separáveis**. Em outros casos, a regra não consegue convergir.
- A **regressão logística** substitui o limiar rígido do perceptron por um limiar suave definido por uma função logística. O gradiente de descida funciona bem, mesmo para dados ruidosos que não são linearmente separáveis.
- As **redes neurais** representam funções não lineares complexas com uma rede de unidades de fronteira linear. A expressão redes neurais de alimentação para a frente pode representar qualquer função, dadas unidades suficientes. O algoritmo de **retropropagação** implementa um gradiente de descida no espaço de parâmetros para minimizar o erro de saída.
- Os **modelos não paramétricos** usam todos os dados para fazer cada previsão, em vez de tentar primeiro resumir os dados com alguns parâmetros. Os exemplos incluem **vizinhos mais próximos** e **regressão ponderada localmente**.
- As **máquinas de vetores de suporte** encontram separadores lineares com **margem máxima** para melhorar o desempenho de generalização do classificador. Os **métodos do kernel** transformam os dados de entrada implicitamente em um espaço de dimensão superior onde pode existir um separador linear, mesmo se os dados originais forem não separáveis.
- Métodos de agrupamento, tais como de **aceleração**, sempre desempenham melhor do que métodos individuais. Em **aprendizagem on-line**, podemos agregar opiniões de especialistas para chegar arbitrariamente perto do melhor desempenho do especialista, mesmo quando a distribuição dos dados estiver em constante mutação.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O Capítulo 1 esboçou a história de investigações filosóficas em aprendizagem indutiva. Guilherme de Ockham[16] (1280-1349), o filósofo mais influente de seu século e um importante colaborador para a epistemologia, a lógica e a metafísica medieval, é considerado o autor de uma declaração denominada "navalha de Ockham" — em latim, *Entia non sunt multiplicanda praeter neces*sitatem ("As entidades não devem ser multiplicadas além da necessidade"). Infelizmente, esse louvável conselho não é encontrado em nenhum lugar em seus escritos exatamente com essas palavras (embora ele tenha dito que "não se deve colocar a maioria sem necessidade"). Em 350 a.C., na *Física*, livro I, capítulo VI, Aristóteles exprimiu um sentimento semelhante: "O mais limitado, se adequado, é sempre preferível."

O primeiro uso notável de árvores de decisão foi em EPAM, o "Elementary Perceiver And Memorizer" (Feigenbaum, 1961), que era uma simulação do conceito humano de aprendizagem. O ID3 (Quinlan, 1979) acrescentou a ideia fundamental de escolher o atributo com entropia máxima; é a

base para o algoritmo de árvore de decisão neste capítulo. A teoria da informação propriamente dita foi desenvolvida por Claude Shannon para auxiliar no estudo de comunicação (Shannon e Weaver, 1949). (Shannon também contribuiu com um dos mais antigos exemplos de aprendizagem de máquina, um camundongo mecânico denominado Theseus que aprendeu a percorrer um labirinto por tentativa e erro.) O método χ^2 de poda de árvore foi descrito por Quinlan (1986). O C4.5, um pacote de árvore de decisão com fins industriais, pode ser encontrado em Quinlan (1993). Uma tradição independente da aprendizagem de árvores de decisão existe na literatura estatística. *Classification and Regression Trees* (Breiman *et al*., 1984), conhecido como "livro CART", é a principal referência.

A **validação cruzada** foi introduzida pela primeira vez por Larson (1931), e de uma forma próxima da que foi apresentada por Stone (1974) e Golub *et al*. (1979). Deve-se a Tikhonov (1963) o procedimento de regularização. Guyon e Elisseeff (2003) apresentaram um artigo de revista dedicada ao problema de seleção de característica. Banko e Brill (2001) e Halevy *et al*. (2009) discutiram as vantagens de utilização de grandes quantidades de dados. Foi Robert Mercer, pesquisador e orador, que disse em 1985: "Não há dados como mais dados." Lyman e Varian (2003) estimam que, em 2002, foram produzidos cerca de 5 exabytes (5×10^{18} bytes) de dados e que a taxa de produção dobra a cada três anos.

A análise teórica de algoritmos de aprendizagem teve início com o trabalho de Gold (1967) em **identificação no limite**. Essa abordagem foi motivada em parte por modelos de descoberta científica a partir da filosofia da ciência (Popper, 1962), mas foi aplicada principalmente ao problema de aprender gramáticas a partir de exemplo de sentenças (Osherson *et al*., 1986).

Enquanto a abordagem de identificação no limite se concentra na convergência ao final, o estudo da **complexidade de Kolmogorov** ou **complexidade algorítmica**, desenvolvido independentemente por Solomonoff (1964) e por Kolmogorov (1965), tenta fornecer uma definição formal para a noção de simplicidade usada na lâmina de Ockham. Para escapar do problema de que a simplicidade depende do modo como a informação é representada, ele propôs que a simplicidade fosse medida pelo comprimento do programa mais curto para uma máquina de Turing universal que reproduz corretamente os dados observados. Embora existam muitas máquinas de Turing universais possíveis e, consequentemente, muitos programas "mais curtos" possíveis, esses programas diferem em comprimento por, no máximo, uma constante que é independente da quantidade de dados. Essa bela ideia, que mostra em essência que qualquer desvio de representação inicial vai eventualmente ser superado pelos próprios dados, só é arruinada pela indecidibilidade do cálculo do comprimento do programa mais curto. Em vez disso, podem ser usadas medidas aproximadas como o **comprimento mínimo de descrição**, ou CMD (Rissanen, 1984), e elas têm produzido excelentes resultados na prática. O texto de Li e Vitanyi (1993) é a melhor fonte para estudo da complexidade de Kolmogorov.

A teoria de aprendizagem computacional — isto é, a teoria de aprendizagem PAC foi inaugurada por Leslie Valiant (1984). Seu trabalho destacou a importância da complexidade computacional e de amostras. Com Michael Kearns (1990), Valiant mostrou que várias classes de conceitos não podem ser aprendidas de forma tratável com a utilização da aprendizagem PAC, embora existam informações suficientes disponíveis nos exemplos. Alguns resultados positivos foram obtidos para classes como listas de decisão (Rivest, 1987).

Uma tradição independente da análise de complexidade de amostra existia em estatística, desde o

trabalho em **teoria de convergência uniforme** (Vapnik e Chervonenkis, 1971). A chamada **dimensão VC** fornece uma medida aproximadamente análoga, embora mais complexa que a medida $\ln|\mathcal{H}|$ obtida a partir da análise PAC. A dimensão VC pode ser aplicada a classes de funções contínuas, às quais a análise PAC padrão não se aplica. A teoria de aprendizagem PAC e a teoria VC foram conectadas primeiro pelos "quatro alemães" (nenhum dos quais é realmente alemão): Blumer, Ehrenfeucht, Haussler e Warmuth (1989).

A regressão linear com a perda de erro quadrático remonta a Legendre (1805) e Gauss (1809), que trabalhavam na previsão de órbitas ao redor do Sol. O uso moderno da regressão multivariada para a aprendizagem de máquina for coberto em textos como de Bishop (2007). Ng (2004) analisou as diferenças entre a regularização L_1 e L_2.

A expressão **função logística** vem de Pierre-François Verhulst (1804-1849), um estatístico que usou a curva para modelar o crescimento da população com recursos limitados, um modelo mais realista do que o crescimento geométrico sem restrições proposto por Thomas Malthus. Verhulst chamou-a de *courbe logistique*, por causa de sua relação com a curva logarítmica. O termo **regressão** é devido a Francis Galton, estatístico do século XIX, primo de Charles Darwin e precussor dos campos da meteorologia, análise de impressões digitais e correlação estatística, que o usou no sentido de regressão até a média. A expresssão **maldição da dimensionalidade** vem de Richard Bellman (1961).

A regressão logística pode ser resolvida com o gradiente de descida ou com o método de Newton-Raphson (Newton, 1671; Raphson, 1690). Uma variante do método de Newton, chamada L-BFGS, algumas vezes é utilizada para problemas de grandes dimensões; o *L* mede a " memória ilimitada", significando que evita a criação de matrizes completas de uma só vez e cria partes delas rapidamente. BFGS são as iniciais dos autores (Byrd *et al*., 1995).

Os modelos de vizinhos mais próximos datam pelo menos de Fix e Hodges (1951) e, desde então, têm sido uma ferramenta-padrão em estatística e reconhecimento de padrões. Em IA, foram popularizados por Stanfill e Waltz (1986), que investigaram métodos para adaptar a distância métrica aos dados. Hastie e Tibshirani (1996) desenvolveram uma maneira de localizar a métrica para cada ponto no espaço, dependendo da distribuição dos dados em torno desse ponto. Gionis *et al*. (1999) introduziram a localidade de hash sensível, que revolucionou a recuperação de objetos similares em espaços de dimensão superior, especialmente em visão computacional. Andoni e Indyk (2006) forneceram recentemente uma pesquisa de métodos HSL e afins.

As ideias por trás das máquinas de kernel vêm de Aizerman *et al*. (1964) (que também introduziram o truque kernel), mas o pleno desenvolvimento da teoria é devido a Vapnik e seus colegas (Boser *et al*., 1992). Os SVMs tornaram-se práticos com a introdução do classificador de margem suave para lidar com dados ruidosos em um artigo que ganhou o 2008 ACM Theory and Practice Award (Cortes e Vapnik, 1995), e o algoritmo Sequential Minimal Optimization (SMO) para resolver de forma eficiente problemas usando programação quadrática SVM (Platt, 1999). Os SVMs provaram ser muito populares e eficazes para tarefas como a categorização de texto (Joachims, 2001), genômica computacional (Cristianini e Hahn, 2007) e processamento de linguagem natural, tais como o reconhecimento de dígitos manuscritos de DeCoste e Schölkopf (2002). Como parte desse processo foram concebidos muitos novos kernels que trabalham com strings, árvores e outros tipos de dados não numéricos. Uma técnica relacionada que também usa o truque do kernel

para representar implicitamente um espaço característico exponencial é o perceptron votado (Freund e Schapire, 1999; Collins e Duffy, 2002). Livros sobre SVMs incluem Cristianini e Shawe-Taylor (2000) e Schölkopf e Smola (2002). Uma exposição mais amigável apareceu em um artigo da *AI Magazin* por Cristianini e Schölkopf (2002). Bengio e LeCun (2007) mostraram algumas das limitações de SVMs e outros locais, métodos não paramétricos para funções de aprendizagem que têm estrutura global, mas não têm a suavidade local.

A aprendizagem de agrupamento é uma técnica cada vez mais popular para melhorar o desempenho de algoritmos de aprendizagem. O **ensacamento** (Breiman, 1996), o primeiro método eficaz, combina hipóteses aprendidas a partir de múltiplos conjunto de dados **bootstrap**, cada um gerado subamostrando o conjunto de dados original. O método de aceleração descrito neste capítulo originou-se com o trabalho teórico de Schapire (1990). O algoritmo ADABOOST foi desenvolvido por Freund e Schapire (1996) e analisado teoricamente por Schapire (2003). Friedman *et al*. (2000) explicaram a aceleração do ponto de vista de um estatístico. A aprendizagem on-line foi abrangida em uma pesquisa realizada pela Blum (1996) e um livro por Cesa-Bianchi e Lugosi (2006). Dredze *et al*. (2008) introduziram a ideia de aprendizagem on-line de confiança ponderada para a classificação: além de manter um peso para cada parâmetro, mantém também uma medida de confiança, de modo que um novo exemplo pode ter grande efeito sobre os recursos que eram raramente vistos antes (e, portanto, tinha baixo nível de confiança) e um pequeno efeito sobre as características comuns que já tinham sido bem estimadas.

A literatura sobre redes neurais é demasiado grande (aproximadamente 150 mil artigos até hoje), para cobrir em detalhe. Cowan e Sharp (1988b, 1988a) pesquisaram o início da história, começando com o trabalho de McCulloch e Pitts (1943). (Como mencionado no Capítulo 1, John McCarthy apontou para o trabalho de Nicolas Rashevsky (1936, 1938) como o primeiro modelo matemático de aprendizagem neural.) Norbert Wiener, um dos pioneiros da cibernética e da teoria de controle (Wiener, 1948), trabalhou com McCulloch e Pitts, e influenciou uma série de jovens investigadores, incluindo Marvin Minsky, que pode ter sido o primeiro a desenvolver uma rede neural em funcionamento em hardware em 1951 (ver Minsky e Papert, 1988, p. ix-x). Turing (1948) escreveu uma pesquisa-relatório intitulada *Intelligent Machinery*, que começa com a sentença "proponho investigar a questão de saber se é possível que as máquinas mostrem comportamento inteligente" e continua a descrever uma arquitetura de rede neural recorrente que chamou de "máquinas desorganizadas tipo B" e uma abordagem para treiná-las. Infelizmente, o relatório foi inédito até 1969, e foi ignorado até recentemente.

Frank Rosenblatt (1957) inventou o "perceptron" moderno e provou o teorema da convergência do perceptron (1960), embora tenha sido prenunciado por ser um trabalho puramente matemático, fora do contexto de redes neurais (Agmon, 1954; Motzkin e Schoenberg, 1954). Foi feito também algum trabalho inicial em redes multicamadas, incluindo **perceptrons de Gamba** (Gamba *et al*., 1961) e **madalines** (Widrow, 1962). As *Máquinas de aprendizagem* (Nilsson, 1965) cobrem grande parte desse trabalho precoce e muito mais. O desaparecimento subsequente dos esforços de pesquisa iniciais do perceptron foi apressado (ou, mais tarde, os autores alegaram, meramente explicado) pelo livro *Perceptrons* (Minsky e Papert, 1969), que lamentou a falta de rigor matemático do campo. O livro apontou que os perceptrons de camada única poderiam representar apenas conceitos linearmente separáveis e notou a falta de algoritmos de aprendizagem efetivos para redes de

múltiplas camadas.

Os artigos em Hinton e Anderson (1981), com base em uma conferência em San Diego em 1979, podem ser considerados como a marca do renascimento do conexionismo. Os dois volumes “PDP” (Parallel Distributed Processing), antologia (Rumelhart *et al.*, 1986a) e um pequeno artigo *Nature* (Rumelhart *et al.*, 1986b) atraíram grande atenção — de fato, o número de artigos sobre “redes neurais”, multiplicou-se por um fator de 200 entre 1980-1984 e 1990-1994. A análise de redes neurais usando a teoria física de vidros de spin magnético (Amit *et al.*, 1985) reforçou as ligações entre a mecânica estatística e a teoria de redes neurais, proporcionando não apenas discernimentos matemáticos úteis, mas também *respeitabilidade*. A técnica de retropropagação foi inventada muito cedo (Bryson e Ho, 1969), mas foi redescoberta várias vezes (Werbos, 1974; Parker, 1985).

A interpretação probabilística de redes neurais tem várias fontes, incluindo Baum e Wilczek (1988) e Bridle (1990). O papel da função sigmoide foi discutido por Jordan (1995). A aprendizagem bayesiana de parâmetros para as redes neurais foi proposta por MacKay (1992) e depois explorada por Neal (1996). A capacidade das redes neurais para representar funções foi investigada por Cybenko (1988, 1989), que mostrou que duas camadas ocultas são suficientes para representar qualquer função e uma única camada é suficiente para representar qualquer função *contínua*. O método do “dano cerebral ótimo” para remover conexões inúteis por LeCun *et al.* (1989) e Sietsma e Dow (1988) mostrou como remover unidades inúteis. O algoritmo de tiling para o crescimento de estruturas maiores é devido a Mézard e Nadal (1989). LeCun *et al.* (1995) pesquisaram uma série de algoritmos de reconhecimento de dígitos escritos à mão. A melhoria das taxas de erro desde então foi relatada por Belongie *et al.* (2002) para encaixe de forma e por DeCoste e Schölkopf (2002) para vetores de suporte virtual. No momento da escrita, a melhor taxa de erro de teste relatado foi de 0,39% em Ranzato *et al.* (2007) utilizando uma rede neural convolucional.

A complexidade da aprendizagem da rede neural tem sido investigada por pesquisadores em teoria da aprendizagem computacional. Os primeiros resultados computacionais foram obtidos por Judd (1990), que mostrou que o problema geral de encontrar um conjunto de pesos consistente com um conjunto de exemplos é NP-completo, mesmo em hipóteses muito restritivas. Um pouco dos resultados de complexidade da primeira amostra foi obtido por Baum e Haussler (1989), que mostraram que o número de exemplos necessários para uma aprendizagem eficaz cresce em cerca de $W \log W$, onde W é o número de pesos.[17] Desde então, uma teoria muito mais sofisticada tem sido desenvolvida (Anthony e Bartlett, 1999), incluindo o resultado importante de que a capacidade de representação de uma rede depende do *tamanho* dos pesos, bem como do seu número, um resultado que não deveria ser surpreendente à luz da nossa discussão de regularização.

O tipo mais popular de rede neural que não abrangemos é a função de base radial, ou FBR, rede. A função de base radial combina uma coleção ponderada de kernels (geralmente gaussianas, é claro) para fazer aproximação de função. As redes FBR podem ser instruídas em duas fases: primeiro, uma abordagem de agrupamento não supervisionado é usado para treinar os parâmetros de gaussianas — as médias e as variâncias — sendo instruídas, como na Seção 20.3.1. Na segunda fase, os pesos relativos das gaussianas são determinados. Esse é um sistema de equações lineares, que sabemos como resolver diretamente. Assim, ambas as fases de treinamento FBR têm uma bela vantagem: a primeira fase não é supervisionada e, portanto, não requer dados de treinamento rotulados, e a

segunda fase, embora supervisionada, é eficiente. Ver Bishop (1995) para mais detalhes.

As **redes recorrentes**, em que as unidades estão ligadas em ciclos, foram mencionadas no capítulo mas não exploradas em profundidade. As **redes de Hopfield** (Hopfield, 1982) provavelmente são a classe de redes recorrentes mais bem entendidas. Usam conexões *bidirecionais* com pesos *simétricos* (ou seja, $w_{i,j} = w_{j,i}$), todas as unidades são unidades de entrada e de saída, a função de ativação g é a função de sinal, e os níveis de ativação só podem ser ±1. A rede de Hopfield funciona como uma **memória associativa**: depois que a rede instrui um conjunto de exemplos, um novo estímulo faz com que ele se estabeleça em um padrão de ativação correspondente ao exemplo do conjunto de treinamento *que mais se assemelhe* ao novo estímulo. Por exemplo, se o conjunto de treinamento consiste em um conjunto de fotografias, e o novo estímulo é um pequeno pedaço de uma das fotografias, os níveis de ativação da rede vão reproduzir a fotografia de onde o pedaço foi tirado. Observe que as fotografias originais não são armazenados separadamente na rede, cada peso é uma codificação parcial de todas as fotografias. Um dos resultados teóricos mais interessantes é que as redes de Hopfield podem armazenar confiavelmente até $0{,}138N$ exemplos de treinamento, onde N é o número de unidades na rede.

As **máquinas de Boltzmann** (Hinton e Sejnowski, 1983, 1986) também utilizam pesos simétricos, mas incluem unidades ocultas. Além disso, utilizam uma função de ativação *estocástica*, tal que a probabilidade de saída sendo 1 é uma função da entrada total ponderada. As máquinas de Boltzmann, portanto, sofrem transições de estado que se assemelham a uma busca de têmpora simulada (veja o Capítulo 4) para a configuração que melhor se aproxima do conjunto de treinamento. Acontece que as máquinas de Boltzmann estão intimamente relacionadas a um caso especial de redes bayesianas avaliadas com um algoritmo de simulação estocástica (veja a Seção 14.5).

No caso das redes neurais, Bishop (1995), Ripley (1996) e Haykin (2008) são os textos principais. O campo da neurociência computacional foi coberto por Dayan e Abbott (2001).

A abordagem adotada neste capítulo foi influenciada pelas excelentes notas do curso de David Cohn, Tom Mitchell, Andrew Moore e Andrew Ng. Existem vários livros didáticos de primeira categoria em aprendizagem de máquina (Mitchell, 1997; Bishop, 2007) e nos campos intimamente aliados e sobrepostos de reconhecimento de padrões (Ripley, 1996;. Duda *et al*., 2001), estatísticas (Wasserman, 2004; Hastie *et al*., 2001), mineração de dados (Hand *et al*., 2001; Witten e Frank, 2005), teoria da aprendizagem computacional (Kearns e Vazirani, 1994; Vapnik, 1998) e teoria da informação (Shannon e Weaver, 1949; MacKay, 2002; Capa e Thomas, 2006). Outros livros se concentram em implementações (Segaran, 2007; Marsland, 2009) e comparações de algoritmos (Michie *et al*., 1994). Pesquisas atuais em aprendizagem de máquina foram publicadas nos procedimentos anuais da International Conference on Machine Learning (ICML) e na conferência sobre Neural Information Processing Systems (NIPS), em *Machine Learning e Journal of Machine Learning Reserach*, e em revistas de tendência atual em IA.

EXERCÍCIOS

18.1 Considere o problema enfrentado por uma criança que aprende a falar e a compreender um idioma. Explique como esse processo se enquadra no modelo geral de aprendizagem. Descreva as

percepções e ações da criança e os tipos de aprendizagem que a criança deve fazer. Descreva as subfunções que o bebê está tentando aprender em termos de entradas e saídas, e os dados de exemplo disponíveis.

18.2 Repita o Exercício 18.1 para o caso de aprender a jogar tênis (ou algum outro esporte com que você esteja familiarizado). Trata-se de uma aprendizagem supervisionada ou de uma aprendizagem por reforço?

18.3 Suponha que geramos um conjunto de treinamento a partir de uma árvore de decisão e depois aplicamos a aprendizagem em árvores de decisão a esse conjunto de treinamento. O algoritmo de aprendizagem vai eventualmente retornar à árvore correta à medida que o tamanho do conjunto de treinamento tender a infinito? Por quê?

18.4 Na construção recursiva de árvores de decisão, às vezes acontece de um conjunto misto de exemplos positivos e negativos permanecer em um nó de folha, mesmo depois que todos os atributos são usados. Vamos supor que temos p exemplos positivos e n exemplos negativos.

a. Mostre que a solução usada por APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO, que escolhe a classificação de maioria, minimiza o erro absoluto sobre o conjunto de exemplos na folha.

b. Mostre que a probabilidade de classe $p/(p + n)$ minimiza a soma de erros quadráticos.

18.5 Suponha que um atributo divida o conjunto de exemplos E em subconjuntos E_k e que cada subconjunto tenha p_k exemplos positivos e n_k exemplos negativos. Mostre que o atributo tem ganho de informações estritamente positivo, a menos que a razão $p_k/(p_k + n_k)$ seja a mesma para todo k.

18.6 Considere o seguinte conjunto de dados composto por três atributos de entrada binários (A_1, A_2 e A_3) e uma saída binária:

Exemplo	A_1	A_2	A_3	Saída y
$\mathbf{x}_1$	1	0	0	0
$\mathbf{x}_2$	1	0	1	0
$\mathbf{x}_3$	0	1	0	0
$\mathbf{x}_4$	1	1	1	1
$\mathbf{x}_5$	1	1	0	1

Utilize o algoritmo na Figura 18.5 para estudar uma árvore de decisão a partir desses dados. Mostre os cálculos feitos para determinar o atributo para dividir em cada nó.

18.7 Um *grafo* de decisão é uma generalização de uma árvore de decisão que permite que os nós (ou seja atributos utilizados para divisões) tenham pais múltiplos, em vez de apenas um único pai. O *grafo* resultante ainda deve ser acíclico. Considere agora a função XOR de *três* atributos de entrada binários, que produzem o valor 1 se e apenas se um número impar dos três atributos de entrada tenha valor 1.

a. Desenhe uma *árvore* de decisão de tamanho mínimo para as três funções XOR de entrada.

b. Desenhe um *grafo* de decisão de tamanho mínimo para as três funções XOR de entrada.

18.8 Este exercício considera χ^2 a poda de árvores de decisão (Seção 18.3.5).

a. Crie um conjunto de dados com dois atributos de entrada, de tal forma que a informação obtida na raiz da árvore para ambos os atributos seja zero, mas haja uma árvore de decisão de profundidade 2 que seja consistente com todos os dados. O que poderia a poda χ^2 fazer nesse conjunto de dados, se aplicada de baixo para cima? Se aplicada de cima para baixo?

b. Modifique a APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO para incluir a poda χ^2. Consulte Quinlan (1986) ou Kears e Mansour (1998) para detalhes.

18.9 O algoritmo APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO padrão descrito no capítulo não manipula casos em que alguns exemplos têm valores de atributos omitidos.

a. Primeiro, precisamos encontrar um modo de classificar tais exemplos, dada uma árvore de decisão que inclua testes sobre os atributos para os quais podem estar faltando valores. Suponha que um exemplo **x** tenha um valor omitido para o atributo *A* e a árvore de decisão efetue testes para *A* em um nó que **x** alcança. Um modo de tratar esse caso é fingir que o exemplo tem todos os valores possíveis para o atributo, mas pesar cada valor de acordo com sua frequência entre todos os exemplos que alcançam esse nó na árvore de decisão. O algoritmo de classificação deve seguir todas as ramificações em qualquer nó para o qual está faltando um valor e multiplicar os pesos ao longo de cada caminho. Escreva um algoritmo de classificação modificado para árvores de decisão que tenha esse comportamento.

b. Agora, modifique o cálculo de ganho de informações para que, em qualquer coleção dada de exemplos *C* em determinado nó na árvore durante o processo de construção, os exemplos com valores omitidos para quaisquer dos atributos restantes recebam valores "como se", de acordo com as frequências desses valores no conjunto *C*.

18.10 Na Seção 18.3.6, observamos que atributos com muitos valores possíveis diferentes podem causar problemas com a medida de ganho. Tais atributos tendem a dividir os exemplos em numerosas classes pequenas ou mesmo classes unitárias, parecendo assim ser altamente relevantes de acordo com a medida de ganho. O critério **razão de ganho** seleciona atributos de acordo com a razão entre seu ganho e seu conteúdo intrínseco de informação, ou seja, a quantidade de informações contidas na resposta à pergunta: "Qual é o valor desse atributo?" Portanto, o critério de razão de ganho tenta medir a eficiência com que um atributo fornece informações sobre a classificação correta de um exemplo. Escreva uma expressão matemática para o conteúdo de informação de um atributo e implemente o critério de razão de ganho em APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO.

18.11 Vamos supor que você esteja executando um experimento de aprendizagem em um novo algoritmo para classificação booleana. Você tem um conjunto de dados que consiste em 100 exemplos positivos e negativos. Você planeja utilizar a validação cruzada com omissão de um e comparar o seu algoritmo com uma função de base, um classificador de maioria simples. (É dado ao classificador de maioria um conjunto de dados de treinamento e, então,este sempre produz a classe que é a maioria no conjunto de treinamento, independentemente da entrada.) Você espera que o classificador de maioria marque cerca de 50% na validação cruzada com omissão de um, mas, para sua surpresa, cada vez ele marca zero. Explique por quê.

18.12 Construa uma *lista de decisão* para classificar os dados a seguir. Selecione os testes para que eles sejam o menor possível (em termos de atributos), quebrando os laços entre os testes com o mesmo número de atributos ao selecionar o que classifica o maior número de exemplos corretamente. Se múltiplos testes tiverem o mesmo número de atributos e classificarem o mesmo número de exemplos, então quebre o laço usando os atributos com os números com índice mais baixo (ou seja, selecione A_1 em vez de A_2).

Exemplo	A_1	A_2	A_3	A_4	y
$\mathbf{x}_1$	1	0	0	0	1
$\mathbf{x}_2$	1	0	1	1	1
$\mathbf{x}_3$	0	1	0	0	1
$\mathbf{x}_4$	0	1	1	0	0
$\mathbf{x}_5$	1	1	0	1	1
$\mathbf{x}_6$	0	1	0	1	0
$\mathbf{x}_7$	0	0	1	1	1
$\mathbf{x}_8$	0	0	1	0	0

18.13 Demonstre que uma lista de decisão possa representar a mesma função que uma árvore de decisão quando utiliza ao máximo tantas regras quanto existam folhas na árvore de decisão para essa função. Dê um exemplo de função representada por uma lista de decisão usando regras estritamente inferiores ao número de folhas em uma árvore de decisão de tamanho mínimo para essa mesma função.

18.14 Este exercício se refere à expressividade de listas de decisão (Seção 18.5).

a. Mostre que listas de decisão podem representar qualquer função booleana se o tamanho dos testes não for limitado.

b. Mostre que, se os testes puderem conter no máximo k literais cada, as listas de decisão poderão representar qualquer função que pode ser representada por uma árvore de decisão de profundidade k.

18.15 Suponha que uma regressão 7-vizinhos-mais-próximos busca retornar $\{7, 6, 8, 4, 7, 11, 100\}$ como o valor de y mais próximo de 7 para dado valor x. Qual é o valor de $\hat{y}$ que minimiza a função perda L_1 desses dados? Há um nome comum em estatística para esse valor como função de valores y, qual é? Responda às mesmas duas perguntas para a função de perda L_2.

18.16 A Figura 18.31 mostrou como um círculo na origem pode estar linearmente separado por mapeamento das características (x_1, x_2) para as duas dimensões (x_1^2, x_2^2). Mas, e se o círculo não estiver localizado na origem? E se for uma elipse, e não um círculo? A equação geral de um círculo (e, portanto, o limite de decisão) é $(x_1 - a)^2 + (x_2 - b)^2 - r^2 = 0$, e a equação geral de uma elipse é c $(x_1 - a)^2 + \text{d}\,(x_2 - b)^2 - 1 = 0$.

a. Expanda a equação do círculo e mostre que os pesos w_i seriam de fronteira de decisão no espaço característico com quatro dimensões (x_1, x_2, x_1^2, x_2^2). Explique por que isso significa que qualquer círculo é linearmente separável nesse espaço.

b. Faça o mesmo para elipses no espaço característico com cinco dimensões $(x_1, x_2, x_1^2, x_2^2, x_1 x_2)$.

18.17 Construa uma máquina de vetor de suporte que calcule a função XOR. Use valores de +1 e −1 (em vez de 1 e 0) para as entradas e saídas, de modo que o exemplo pareça ([−1, 1], 1) ou ([−1, −1], −1). Faça o mapeamento da entrada $[x_1, x_2]$ em um espaço constituído por x_1 e $x_1 x_2$. Desenhe os quatro pontos de entrada nesse espaço e o separador de margem máximo. Qual é a margem? Agora, desenhe a linha que separa de volta no espaço de entrada original euclidiano.

18.18 Considere um algoritmo de aprendizagem por agrupamento que use a votação de maioria simples entre K hipóteses aprendidas. Suponha que cada hipótese tenha erro $\in$ e que os erros cometidos por cada hipótese sejam independentes dos erros das outras hipóteses. Calcule uma fórmula para o erro do algoritmo de conjunto em termos de K e $\in$, e avalie essa fórmula para os casos em que $K = 5$, 10 e 20, e $\in = 0{,}1$, 0,2 e 0,4. Se a suposição de independência for removida, será possível o erro de conjunto ser *pior* que $\in$?

18.19 Construa uma rede neural manualmente que calcule a função XOR de duas entradas. Certifique-se de especificar que tipo de unidades está usando.

18.20 Lembre-se do Capítulo 18 que existem 2^{2n} funções booleanas distintas de n entradas. Como muitas delas são representáveis por um perceptron de limiar?

18.21 A Seção 18.6.4 observou que a saída da função logística poderia ser interpretada como uma *probabilidade p* atribuída pelo modelo para a proposição de que $f(\mathbf{x}) = 1$; a probabilidade de que $f(\mathbf{x}) = 0$ é, portanto, 1 − p. Escreva a probabilidade p como uma função de **x** e calcule a derivada de $\log p$ com relação a cada peso w_i. Repita o processo para $\log (1 - p)$. Esses cálculos dão uma regra de aprendizagem para minimizar a função de perda de probabilidade log negativo para uma hipótese probabilística. Comente sobre qualquer semelhança com outras regras de aprendizagem no capítulo.

18.22 Suponha que você tenha uma rede neural com funções de ativação linear. Ou seja, para cada unidade, a saída é alguma constante c vezes a soma ponderada das entradas.

a. Suponha que a rede tenha uma camada oculta. Para uma atribuição **w** dada aos pesos, escreva as equações para o valor das unidades na camada de saída em função de **w** e na camada de entrada **x**, sem qualquer menção explícita da saída da camada oculta. Mostre que existe uma rede sem unidades ocultas que calcula a mesma função.

b. Repita o cálculo da parte (a), mas dessa vez faça-o para uma rede com qualquer número de camadas ocultas.

c. Suponha que uma rede com uma camada oculta e funções de ativação linear tenha n nós de entrada e saída e h nós ocultos. Qual efeito a transformação na parte (a) de uma rede sem camadas ocultas tem sobre o número total de pesos? Discuta em particular o caso $h \ll n$.

18.23 Suponha que um conjunto de treinamento contenha apenas um único exemplo com 100 repetições. Em 80 dos 100 casos, o único valor de saída é 1; nos outros 20 é 0. O que a rede de retro

propagação prediz para esse exemplo, assumindo que foi treinada e atinge um ótimo global? (Dica: para encontrar o ótimo global, derive a função erro e a defina como zero.)

18.24 A rede neural cujo desempenho de aprendizagem é medido na Figura 18.25 tem quatro nós ocultos. Esse número foi escolhido arbitrariamente. Use o método de validação cruzada para encontrar o melhor número de nós ocultos.

18.25 Considere o problema de separar N pontos de dados em exemplos positivos e negativos usando um separador linear. Por certo, isso sempre pode ser feito para $N = 2$ pontos em uma linha de dimensão $d = 1$, independentemente de como os pontos estão rotulados ou de onde eles estão localizados (a menos que os pontos estejam no mesmo lugar).

a. Mostre que sempre pode ser feito para $N = 3$ pontos em um plano de dimensão $d = 2$, a menos que sejam colineares.

b. Mostre que sempre pode ser feito para $N = 4$ pontos em um plano de dimensão $d = 2$.

c. Mostre que sempre pode ser feito para $N = 4$ pontos em um espaço de dimensão $d = 3$, a menos que sejam coplanares.

d. Mostre que sempre pode ser feito para $N = 5$ pontos em um espaço de dimensão $d = 3$.

e. O estudante ambicioso pode querer provar que N pontos em posição geral (mas não $N + 1$) são linearmente separáveis em um espaço de dimensão $N - 1$.

---

[1] Uma observação quanto à notação: exceto onde observado, usaremos j para indexar os N exemplos; x_j será sempre a entrada e y_j a saída. Nos casos em que a entrada é especificamente um vetor de valores de atributos (começando na Seção 18.3), vamos usar x_j para o j-ésimo exemplo e i para indexar os n atributos de cada exemplo. Os elementos de $\mathbf{x}_j$ são escritos como $x_{j,1}, x_{j,2}, \ldots, x_{j,n}$.

[2] O ganho será estritamente positivo, exceto para o caso improvável em que todas as proporções forem *exatamente* as mesmas. (Veja o Exercício 18.5.)

[3] Gauss mostrou que, se os valores y_j normalmente têm ruído distribuído, os valores mais prováveis de w_1 e w_0 são obtidos através da minimização da soma dos quadrados dos erros.

[4] Com algumas ressalvas: a função de perda L_2 será adequada quando houver ruído normalmente distribuído, que seja independente de x; todos os resultados dependem da suposição de estacionaridade etc.

[5] O leitor pode consultar o Apêndice A para um breve resumo da álgebra linear.

[6] Talvez seja confuso que L_1 e L_2 sejam usados tanto para as funções de perda como de regularização. Eles não precisam ser usados em pares: você pode usar a perda L_2 com a regularização L_1 ou vice-versa.

[7] Tecnicamente é necessário que $\sum_{t=1}^{\infty} \alpha(t) = \infty$ e $\sum_{t=1}^{\infty} \alpha^2(t) < \infty$. O decaimento a($t$) = $O(1/t)$ satisfaz essas condições.

[8] Uma observação sobre a notação: nesta seção, fomos forçados a suspender nossas convenções usuais. Atributos de entrada ainda são indexados por i, de modo que uma ativação "externa" a_i é dada pela entrada x_i, mas o índice j refere-se a unidades internas em vez de exemplos. Em toda esta seção, as derivações matemáticas dizem respeito a um único exemplo genérico $\mathbf{x}$, omitindo os somatórios usuais de exemplos para obter resultados para o conjunto de dados.

[9] A prova é complexa, mas o ponto principal é que o número necessário de unidades ocultas cresce exponencialmente com o número de entradas. Por exemplo, são necessárias $2^n/n$ unidades ocultas para codificar todas as funções booleanas de n entradas.

[10] Tem-se observado que as redes muito grandes fazem bem a generalização, *desde que os pesos sejam mantidos pequenos.* Essa restrição mantém os valores de ativação na região *linear* da função sigmoide $g(x)$ onde x é próximo de zero. Isso, por sua vez, significa que a rede se comporta como uma função linear (Exercício 18.22), com muito menos parâmetros.

[11] O leitor pode perceber que poderíamos ter usado apenas f_1 e f_2, mas o mapeamento 3-D ilustra melhor a ideia.

[12] Esse uso da "função kernel" é ligeiramente diferente dos kernels em regressão ponderada localmente. Alguns kernels SVM são métricas de distância, mas nem todos são.

[13] Aqui, "razoável" significa que a matriz $\mathbf{K}_{jk} = K(\mathbf{x}_j, \mathbf{x}_k)$ é definida positiva.

[14] Para algoritmos de aprendizagem em que isso não é possível, podemos criar um conjunto de treinamento replicado onde o *i*-ésimo exemplo aparece wj vezes usando a aleatoriedade para tratar pesos fracionários.

[15] Consulte Blum (1996) para a demonstração.

[16] O nome é muitas vezes grafado como "Occam", talvez a partir do francês, Guillaume d'Occam.

[17] Isso confirmou aproximadamente a "regra do tio Bernie". A regra foi denominada devido a Bernie Widrow, que recomendou usar cerca de 10 vezes tanto os exemplos como os pesos.

CAPÍTULO 19

Conhecimento em aprendizagem

Em que examinamos o problema de aprendizagem quando você já sabe algo.

Em todas as abordagens para estudo da aprendizagem descritas nos capítulos anteriores, a ideia é construir uma função que tem o comportamento de entrada/saída observada nos dados. Em cada caso, os métodos de aprendizagem podem ser entendidos como a busca em um espaço de hipóteses para encontrar uma função apropriada, começando apenas por uma suposição muito básica sobre a forma da função, como "polinômio de segundo grau" ou "árvore de decisão" e, talvez, a preferência por uma hipótese mais simples. Isso significa dizer que, antes de poder aprender algo novo, primeiro você deve esquecer (quase) tudo o que sabe. Neste capítulo, estudaremos métodos de aprendizagem que podem tirar proveito do **conhecimento *a priori*** sobre o mundo. Na maioria dos casos, o conhecimento *a priori* é representado como teorias lógicas gerais de primeira ordem; desse modo, pela primeira vez, juntamos o trabalho sobre representação do conhecimento e o de aprendizagem.

19.1 UMA FORMULAÇÃO LÓGICA DA APRENDIZAGEM

O Capítulo 18 definiu a aprendizagem indutiva pura como um processo de encontrar uma hipótese que concorde com os exemplos observados. Aqui, especializaremos essa definição para o caso em que a hipótese é representada por um conjunto de sentenças lógicas. Descrições de exemplos e classificações também serão sentenças lógicas, e um novo exemplo poderá ser classificado deduzindo-se uma sentença de classificação a partir da hipótese e da descrição do exemplo. Essa abordagem permite a construção incremental de hipóteses, uma sentença de cada vez. Ela também permite o conhecimento *a priori* porque sentenças que já são conhecidas podem ajudar na classificação de novos exemplos. A princípio, a formulação lógica de aprendizagem talvez pareça muito trabalho extra, mas ela serve para esclarecer muitas das questões relacionadas à aprendizagem. Além disso, ela nos permite ir muito além dos métodos de aprendizagem simples do Capítulo 18, usando todo o poder de inferência lógica a serviço da aprendizagem.

19.1.1 Exemplos e hipóteses

Vimos no Capítulo 18 o problema de aprendizagem do restaurante: aprender uma regra para decidir se devemos esperar por uma mesa. Os exemplos foram descritos por **atributos** como *Alternativa*, *Bar*, *Sex/Sáb*, e assim por diante. Em uma configuração lógica, um exemplo é um objeto descrito por uma sentença lógica; os atributos se tornam predicados unários. Vamos chamar genericamente o *i*-ésimo exemplo de X_i. Assim, o primeiro exemplo da Figura 18.3 é descrito pelas sentenças:

$$Alternativa(X_1) \land \neg Bar(X_1) \land \neg Sex/Sáb(X_1) \land Faminto(X_1) \land \ldots$$

Usaremos a notação $D_i(X_i)$ para fazer referência à descrição de X_i, onde D_i pode ser qualquer expressão lógica que recebe um único argumento. A classificação do exemplo é dado por um literal utilizando o predicado objetivo, nesse caso

$$VaiEsperar(X_1) \quad \text{ou} \quad \neg VaiEsperar(X_1).$$

O conjunto de treinamento completo é então simplesmente a conjunção de todas as descrições de exemplos e literais objetivos.

O objetivo da aprendizagem indutiva em geral é encontrar uma hipótese que classifique os exemplos bem e generalize bem para novos exemplos. Aqui estamos preocupados com hipóteses expressas em lógica; cada hipótese h_j terá a forma

$$\forall x\ Objetivo(x) \Leftrightarrow C_j(x),$$

onde $C_j(x)$ é uma definição candidata — alguma expressão envolvendo os predicados dos atributo. Por exemplo, uma árvore de decisão pode ser interpretada como uma expressão lógica dessa fórmula. Desse modo, a Figura 18.6 expressa a definição lógica a seguir (que chamaremos de h_r para referência futura):

$$\begin{aligned} \forall r\ VaiEsperar(r) \Leftrightarrow\ & Clientes(r, Alguns) \\ \lor\ & Clientes(r, Cheio) \land Faminto(r) \land Tipo(r, Francês) \\ \lor\ & Clientes(r, Cheio) \land Faminto(r) \land Tipo(r, Tailandês) \\ & \quad \land Sex/Sáb(r) \\ \lor\ & Clientes(r, Cheio) \land Faminto(r) \land Tipo(r, Hambúrguer)\ . \end{aligned} \tag{19.1}$$

Cada hipótese prevê que certo conjunto de exemplos — ou seja, aqueles que satisfazem à sua definição candidata — será o conjunto de exemplos do predicado objetivo. Esse conjunto é chamado **extensão** do predicado.

Duas hipóteses com extensões diferentes são então logicamente inconsistentes uma em relação à outra porque elas discordam em suas previsões por pelo menos um exemplo. Se tiverem a mesma extensão, elas serão logicamente equivalentes.

O espaço de hipóteses $\mathcal{H}$ é o conjunto de todas as hipóteses $\{h_1, \ldots, h_n\}$ que o algoritmo de aprendizagem foi projetado para considerar. Por exemplo, o algoritmo APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO pode considerar qualquer hipótese de árvore de decisão definida em termos dos atributos fornecidos; portanto, seu espaço de hipóteses consiste em todas essas árvores de

decisão. Presumivelmente, o algoritmo de aprendizagem acredita que uma das hipóteses seja correta; isto é, ele acredita na sentença

$$h_1 \vee h_2 \vee h_3 \vee \ldots \vee h_n. \quad (19.2)$$

À medida que os exemplos chegam, as hipóteses que não são **consistentes** com os exemplos podem ser eliminadas. Vamos examinar essa noção de consistência com mais atenção. É óbvio que, se a hipótese h_j é consistente com o conjunto de treinamento inteiro, ela tem de ser consistente com cada exemplo no conjunto de treinamento. O que significaria para ela o fato de ser inconsistente com um exemplo? Isso pode acontecer de duas maneiras:

- Um exemplo pode ser **falso negativo** para a hipótese se a hipótese afirmar que ele deve ser negativo, mas de fato ele for positivo. Então, o novo exemplo X_{13} descrito por

  $$Clientes(X_{13}, Cheio) \wedge \neg Faminto(X_{13}) \wedge \ldots \wedge VaiEsperar(X_{13})$$

  seria um falso negativo para a hipótese h_r dada anteriormente. A partir de h_r e da descrição do exemplo, podemos deduzir tanto $VaiEsperar(X_{13})$, que é a afirmação do exemplo, quanto $\neg VaiEsperar(X_{13})$, o que a hipótese prevê. A hipótese e o exemplo são então logicamente inconsistentes.
- Um exemplo pode ser **falso positivo** para a hipótese se a hipótese afirmar que ele deve ser positivo, mas de fato ele for negativo.[1]

Se um exemplo é falso positivo ou falso negativo para uma hipótese, então o exemplo e a hipótese são logicamente inconsistentes um com o outro. Supondo-se que o exemplo seja uma observação correta do fato, a hipótese pode ser eliminada. Em termos lógicos, isso é exatamente análogo à regra de resolução de inferência (veja o Capítulo 9), pela qual a disjunção de hipóteses corresponde a uma cláusula e o exemplo corresponde a um literal que se resolve em relação a um dos literais na cláusula. Um sistema de inferência lógica comum poderia, em princípio, aprender a partir do exemplo, eliminando uma ou mais hipóteses. Vamos supor que o exemplo seja denotado pela sentença I_1 e que o espaço de hipóteses seja $h_1 \vee h_2 \vee h_3 \vee h_4$. Então, se I_1 é inconsistente com h_2 e h_3, o sistema de inferência lógica pode deduzir o novo espaço de hipóteses $h_1 \vee h_4$.

Assim, podemos caracterizar a aprendizagem indutiva em uma configuração lógica como um processo de eliminação gradual de hipóteses que são inconsistentes com os exemplos, reduzindo as possibilidades. Como o espaço de hipóteses normalmente é vasto (ou até mesmo infinito, no caso da lógica de primeira ordem), não recomendamos tentar construir um sistema de aprendizagem usando a prova de teoremas baseada na resolução e uma enumeração completa do espaço de hipóteses. Em vez disso, descreveremos duas abordagens que encontram hipóteses logicamente consistentes com muito menos esforço.

19.1.2 Busca da melhor hipótese corrente

A ideia por trás da busca da **melhor hipótese corrente** é manter uma única hipótese e ajustá-la à medida que chegam novos exemplos, a fim de manter a consistência. O algoritmo básico foi descrito por John Stuart Mill (1843) e talvez ainda tenha aparecido bem antes disso.

Vamos supor que tenhamos alguma hipótese como h_r, à qual nos afeiçoamos. Como cada novo exemplo é consistente, não precisamos fazer nada. Então, surge um exemplo falso negativo, X_{13}. O que fazemos? A Figura 19.1(a) mostra esquematicamente h_r como uma região: tudo o que está dentro do retângulo faz parte da extensão de h_r. Os exemplos que realmente foram vistos até agora são mostrados como "+" ou "–" e vemos que h_r divide corretamente em categorias todos os exemplos como exemplos positivos ou negativos de *VaiEsperar*. Na Figura 19.1(b), um novo exemplo (dentro do círculo) é um falso negativo: a hipótese afirma que ele deve ser negativo, mas, na verdade, ele é positivo. A extensão da hipótese deve ser aumentada para incluí-lo. Isso é chamado **generalização**; uma generalização possível é mostrada na Figura 19.1(c). Então, na Figura 19.1(d), vemos um falso positivo: a hipótese afirma que o novo exemplo (no círculo) deve ser positivo, mas na realidade ele é negativo. A extensão da hipótese deve ser diminuída para excluir o exemplo. Isso se chama **especialização**; na Figura 19.1(e), vemos uma especialização possível da hipótese. As relações "mais geral que" e "mais específica que" entre hipóteses fornecem a estrutura lógica no espaço de hipóteses que tornam a busca eficiente possível.

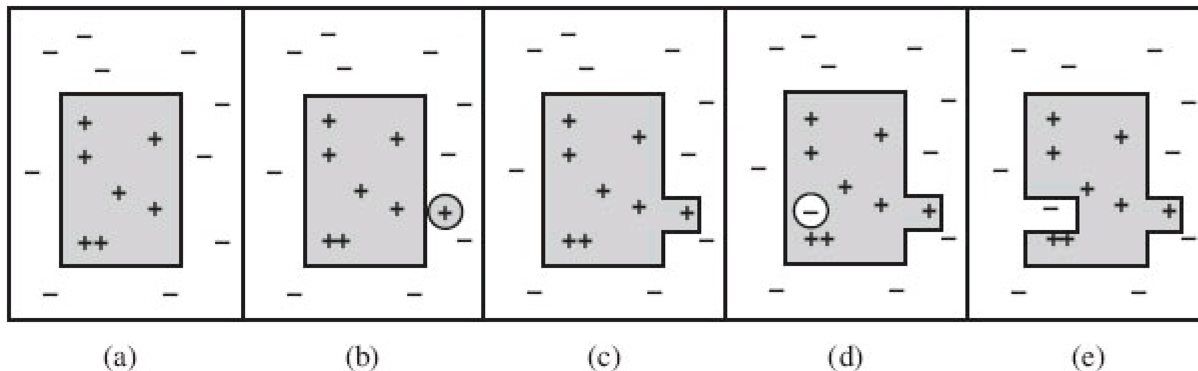


Figura 19.1 (a) Hipótese consistente. (b) Falso negativo. (c) A hipótese é generalizada. (d) Falso positivo. (e) A hipótese é especializada.

Agora, podemos especificar o algoritmo APRENDIZAGEM-MELHOR-CORRENTE, mostrado na Figura 19.2. Note que, toda vez que consideramos a possibilidade de generalizar ou especializar a hipótese, devemos verificar a consistência com os outros exemplos porque aumento/diminuição arbitrário(a) na extensão poderia incluir/excluir exemplos negativos/positivos vistos anteriormente.

```
função APRENDIZAGEM-MELHOR-CORRENTE(exemplos, h) retorna uma hipótese ou falha
  se exemplos vazio então
    retornar h
  e ← PRIMEIRO(exemplos)
  se e é consistente com h então
    retornar APRENDIZAGEM-MELHOR-CORRENTE (RESTO(exemplos), h)
  senão se e é falso positivo de h então
    para cada h' em especializações de h consistente com exemplos vistos até agora faça
```

h'' ← APRENDIZAGEM-MELHOR-CORRENTE (RESTO(*exemplos), h'*)
se *h''* ≠ falha **então retornar** *h''*
senão se *e* é falso negativo para *h* **então**
para cada *h'* **em** generalizações de *h* consistente com *exemplos* vistos até agora **faça**
h''← APRENDIZAGEM-MELHOR-CORRENTE (RESTO(*exemplos), h'*)
se *h''* ≠ falha **então retornar** *h''*
retornar *falha*

Figura 19.2 Algoritmo de aprendizagem de melhor hipótese corrente. Ele busca uma hipótese consistente que mais se adapte aos exemplos e retrocede quando não é possível encontrar nenhuma especialização/generalização consistente. Para iniciar o algoritmo, qualquer hipótese pode ser aceitável; será especializada ou generalizada na medida da necessidade.

Definimos generalização e especialização como operações que mudam a *extensão* de uma hipótese. Agora, precisamos determinar exatamente como elas podem ser implementadas sob a forma de operações sintáticas que mudam a definição candidata associada à hipótese de forma que um programa possa executá-las. Isso é feito observando-se primeiro que generalização e especialização também são relacionamentos *lógicos* entre hipóteses. Se a hipótese h_1 com a definição C_1 é uma generalização da hipótese h_2 com a definição C_2, devemos ter:

$$\forall x\ C_2(x) \Rightarrow C_1(x)\ .$$

Então, para construir uma generalização de h_2, precisamos simplesmente encontrar uma definição de C_1 que seja logicamente implicada por C_2. Isso é feito com facilidade. Por exemplo, se $C_2(x)$ é *Alternativa*(x) ∧ *Clientes* (x, *Alguns*), então uma generalização possível é dada por $C_1(x)$ ≡ *Clientes*(x, *Alguns*). Isso se chama **descarte de condições**. Intuitivamente, o descarte de condições gera uma definição mais fraca e, por conseguinte, permite um conjunto maior de exemplos positivos. Existem várias outras operações de generalização, dependendo da linguagem que está sendo utilizada. De modo semelhante, podemos especializar uma hipótese adicionando condições extras à sua definição candidata ou removendo disjuntos a partir de uma definição disjuntiva. Vamos ver como isso funciona no exemplo de restaurante, usando os dados da Figura 18.3.

- O primeiro exemplo X_1 é positivo. O atributo *Alternativa*(X_1) é verdadeiro e, assim, seja a hipótese inicial

$$h_1 : \forall x\ VaiEsperar(x) \Leftrightarrow Alternativa(x).$$

- O segundo exemplo, X_2, é negativo. h_1 prevê que ele deve ser positivo; então, trata-se de um falso positivo. Assim, precisamos especializar h_1. Isso pode ser feito adicionando-se uma condição extra que eliminará X_2, enquanto continua a classificar X_1 como positivo. Uma possibilidade é

$$h_2 : \forall x\ VaiEsperar(x) \Leftrightarrow Alternativa(x) \wedge Clientes(x, Alguns).$$

- O terceiro exemplo, X_3, é positivo. h_2 prevê que ele deve ser negativo e, assim, ele é um falso negativo. Portanto, precisamos generalizar h_2. Descartamos a condição *Alternativa*, gerando:

$$h_3 : \forall x \ VaiEsperar(x) \Leftrightarrow Clientes(x, Alguns).$$

- O quarto exemplo, X_4, é positivo. h_3 prevê que ele deve ser negativo; então, trata-se de um falso negativo. Portanto, precisamos generalizar h_3. Não podemos descartar a condição de *Clientes* porque isso produziria uma hipótese totalmente inclusiva que seria inconsistente com X_2. Uma possibilidade é adicionar um disjunção:

$$h_4 : \forall x \ VaiEsperar(x) \Leftrightarrow Clientes(x, Alguns) \lor (Clientes(x, Cheio) \land Sex/Sáb(x)).$$

Com isso, a hipótese já está começando a parecer razoável. É evidente que existem outras possibilidades consistentes com os quatro primeiros exemplos; aqui estão duas delas:

$$h'_4 : \forall x \ VaiEsperar(x) \Leftrightarrow \neg EsperaEstimada(x, 30\text{-}60).$$
$$h''_4 : \forall x \ VaiEsperar(x) \Leftrightarrow Clientes(x, Alguns) \lor (Clientes(x, Cheio) \land EsperaEstimada(x, 10\text{-}30)).$$

O algoritmo APRENDIZAGEM-MELHOR-CORRENTE é descrito de forma não determinística porque, em qualquer ponto, talvez haja várias especializações ou generalizações possíveis que podem ser aplicadas. As escolhas que são feitas não levarão necessariamente à hipótese mais simples e podem resultar em uma situação irrecuperável, na qual nenhuma modificação simples da hipótese será consistente com todos os dados.

Em tais casos, o programa deve retroceder a um ponto de escolha anterior.

O algoritmo APRENDIZAGEM-MELHOR-CORRENTE e suas variantes foram usados em muitos sistemas de aprendizagem de máquina, começando com o programa de "aprendizagem de arco" de Patrick Winston (1970). Entretanto, com grande número de instâncias e espaço extenso, surgem algumas dificuldades:

1. A verificação de todas as instâncias anteriores repetidamente para cada modificação é muito dispendiosa.
2. O processo de busca pode envolver muito retrocesso. Como vimos no Capítulo 18, o espaço de hipóteses pode ser um lugar duplo exponencialmente grande.

19.1.3 Busca de compromisso mínimo

O retrocesso surge porque a abordagem de melhor hipótese corrente tem de *escolher* uma hipótese específica como sua melhor suposição, embora ainda não tenha dados suficientes para ter certeza da escolha. Em vez disso, podemos contornar todas as hipóteses que são consistentes com todos os dados até agora, e somente essas. Cada nova instância não terá nenhum efeito ou se livrará de algumas das hipóteses. Lembre-se de que o espaço de hipóteses original pode ser visto como uma

sentença disjuntiva:

$h_1 \lor h_2 \lor h_3 \ldots \lor h_n$.

À medida que descobrimos que várias hipóteses são inconsistentes com os exemplos, essa disjunção é reduzida, retendo apenas as hipóteses não eliminadas. Supondo-se que o espaço de hipóteses original contenha de fato a resposta correta, a disjunção reduzida ainda deverá conter a resposta correta porque apenas hipóteses incorretas foram removidas. O conjunto de hipóteses restante é chamado **espaço de versão**, e o algoritmo de aprendizagem (esboçado na Figura 19.3) é chamado algoritmo de aprendizagem de espaço de versão (e também de algoritmo de **eliminação de candidata**).

função APRENDIZAGEM-ESPAÇO-DE-VERSÃO(*exemplos*) **retorna** um espaço de versão
  variáveis locais: *V*, o espaço de versão: o conjunto de todas as hipóteses
  V ← o conjunto de todas as hipóteses
  para cada exemplo *e* em *exemplos* **faça**
    se *V* não é vazio **então** *V* ← ATUALIZAR-ESPAÇO-DE-VERSÃO(*V*, *e*)
  retornar *V*

---

função ATUALIZAR-ESPAÇO-DE-VERSÃO(*V*, *e*) **retorna** um espaço de versão atualizado
  $V \leftarrow \{h \in V : h$ é consistente com $e\}$

Figura 19.3 Algoritmo de aprendizagem de espaço de versão. Ele encontra um subconjunto de *V* que é consistente com os *exemplos*.

Uma propriedade importante dessa abordagem é que ela é *incremental*: nunca se tem de voltar e reexaminar os exemplos antigos. Todas as hipóteses são garantidas de já estar consistentes com eles.

Porém, existe um problema óbvio. Já dissemos que o espaço de hipóteses é enorme; então, como poderíamos escrever essa enorme disjunção?

A analogia simples a seguir é muito útil. Como representar todos os números reais entre 1 e 2? Afinal, existe uma quantidade infinita deles! A resposta é usar uma representação de intervalo que só especifica os limites do conjunto: [1,2]. Ela funciona porque temos uma *ordenação* sobre os números reais.

Também temos uma ordenação sobre o espaço de hipóteses, isto é, generalização/especialização. Essa é uma ordenação parcial, o que significa que cada limite não será um ponto, mas um conjunto de hipóteses chamado **conjunto limite**. O detalhe interessante é que podemos representar o espaço de versão inteiro usando apenas dois conjuntos limite: um limite mais geral (o **conjunto G**) e um limite mais específico (o **conjunto S**). *Tudo o que estiver entre eles tem a garantia de ser consistente com os exemplos*. Antes de provarmos esse fato, vamos recapitular:

- O espaço de versão corrente é o conjunto de hipóteses consistentes com todos os exemplos até agora. Ele é representado pelo conjunto S e pelo conjunto G, cada um dos quais é um conjunto de hipóteses.

- Todo elemento do conjunto S é consistente com todas as observações até agora, e não existe nenhuma hipótese consistente que seja mais específica.
- Todo elemento do conjunto G é consistente com todas as observações até agora, e não existe nenhuma hipótese consistente que seja mais geral.

Queremos que o espaço de versão inicial (antes de quaisquer exemplos serem vistos) represente todas as hipóteses possíveis. Fazemos isso configurando o conjunto G para conter *Verdadeiro* (a hipótese que contém tudo) e o conjunto S para conter *Falso* (a hipótese cuja extensão está vazia).

A Figura 19.4 mostra a estrutura geral da representação de conjunto limite do espaço de versão. Para mostrar que a representação é suficiente, precisamos das duas propriedades a seguir:

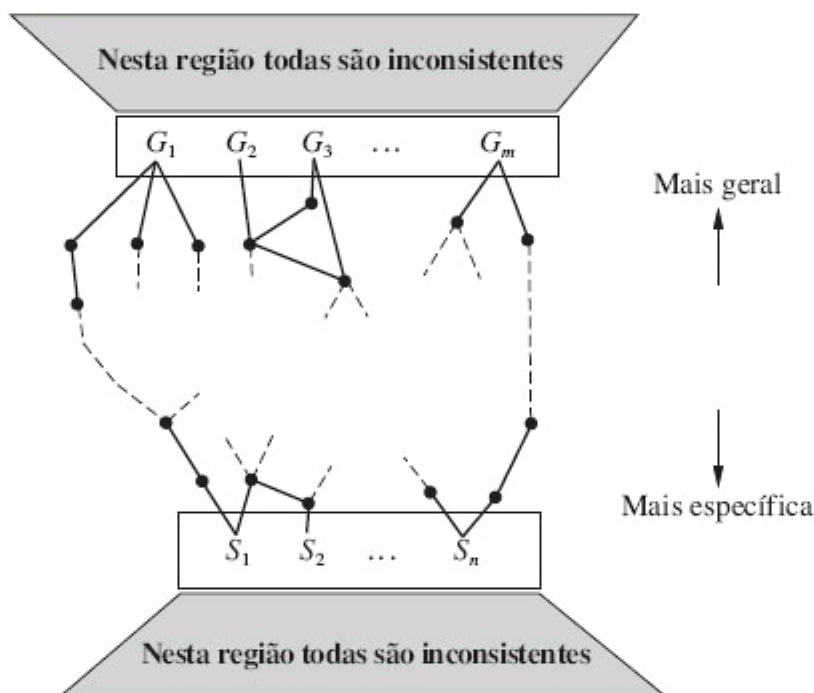


Figura 19.4 O espaço de versão contém todas as hipóteses consistentes com os exemplos.

1. Toda hipótese consistente (além daquelas dos conjuntos de limites) é mais específica que algum elemento do conjunto G e mais geral que algum elemento do conjunto S (isto é, não existe nenhuma hipótese "errante" que tenha ficado de fora). Isso decorre diretamente das definições de *S* e *G*. Se houvesse uma hipótese errante *h*, ela não poderia ser mais específica que qualquer elemento de *G* e, nesse caso, pertenceria a *G*; ou, então, ela não seria mais geral que qualquer elemento de *S* e, nesse caso, estaria em *S*.
2. Toda hipótese mais específica que algum elemento do conjunto G e mais geral que algum elemento do conjunto S é uma hipótese consistente (isto é, não existe nenhum "buraco" entre os limites). Qualquer *h* entre *S* e *G* deve rejeitar todos os exemplos negativos rejeitados por cada elemento de *G* (porque ela é mais específica) e deve aceitar todos os exemplos positivos aceitos por qualquer elemento de *S* (porque é mais geral). Desse modo, *h* deve concordar com todos os exemplos e, portanto, não pode ser inconsistente. A Figura 19.5 mostra a situação: não existe nenhum exemplo conhecido fora de *S* mas dentro de *G* e, assim, qualquer hipótese no intervalo deve ser consistente.

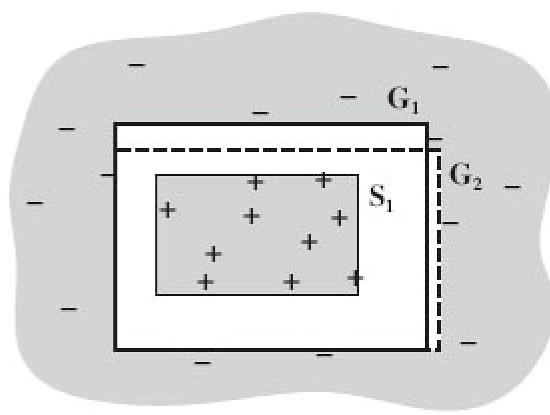


Figura 19.5 Extensões dos elementos de *G* e *S*. Nenhum exemplo conhecido reside entre os dois conjuntos de limites.

Então, mostramos que, se *S* e *G* são mantidos de acordo com suas definições, eles devem fornecer uma representação satisfatória do espaço de versão. O único problema que resta é o de *atualizar S* e *G* para um novo exemplo (o trabalho da função ATUALIZAR-ESPAÇO-DE-VERSÃO). Isso pode parecer bastante complicado a princípio; porém, a partir das definições e com a ajuda da Figura 19.4, não é tão difícil reconstruir o algoritmo.

Precisamos nos preocupar com os elementos S_i e G_i dos conjuntos S e G. Para cada um, a nova instância pode ser um falso positivo ou um falso negativo.

1. Falso positivo para S_i: isso significa que S_i é muito geral, mas não existe nenhuma especialização consistente de S_i (por definição) e, assim, nós o retiramos do conjunto S.
2. Falso negativo para S_i: isso significa que S_i é muito específico, então nós o substituímos por todas as suas generalizações imediatas, desde que elas sejam mais específicas que algum elemento de *G*.
3. Falso positivo para G_i: isso significa que G_i é muito geral, então nós o substituímos por todas as suas especializações imediatas, desde que elas sejam mais gerais que algum elemento de *S*.
4. Falso negativo para G_i: isso significa que G_i é muito específico, mas não existe nenhuma generalização consistente de G_i (por definição) e, assim, nós o retiramos do conjunto G.

Continuamos com essas operações para cada nova instância até ocorrer uma destas três situações:

1. Haver exatamente um conceito restante no espaço de versão e, nesse caso, nós o retornaremos como a única hipótese.
2. O espaço de versão *entra em colapso* — S ou G fica vazio, indicando que não existem hipóteses consistentes para o conjunto de treinamento. Esse é o mesmo caso de falha da versão simples do algoritmo de árvore de decisão.
3. Esgotamos os exemplos e temos várias hipóteses que ainda restavam no espaço de versão. Isso significa que o espaço de versão representa uma disjunção de hipóteses. Para qualquer novo exemplo, se todas as disjunções concordarem, poderemos retornar sua classificação do exemplo. Se elas discordarem, uma possibilidade será realizar a votação de maioria.

Deixamos como exercício a aplicação do algoritmo APRENDIZAGEM-ESPAÇO-DE-VERSÃO aos dados do restaurante.

Há algumas desvantagens importantes na abordagem de espaço de versão:

- Se o domínio contiver ruído ou atributos insuficientes para classificação exata, o espaço de versão sempre entrará em colapso.
- Se permitirmos disjunção ilimitada no espaço de hipóteses, o conjunto S sempre conterá uma única hipótese mais específica, ou seja, a disjunção das descrições dos exemplos positivos vistos até o momento. De modo semelhante, o conjunto G conterá apenas a negação da disjunção das descrições dos exemplos negativos.
- Para alguns espaços de hipóteses, o número de elementos no conjunto S do conjunto G pode crescer exponencialmente no número de atributos, embora existam algoritmos de aprendizagem eficientes para esses espaços de hipóteses.

Até agora, não foi encontrada nenhuma solução completamente bem-sucedida para o problema de ruído. O problema de disjunção pode ser tratado permitindo apenas formas limitadas de disjunção ou incluindo-se uma **hierarquia de generalização** de predicados mais gerais. Por exemplo, em vez de usar a disjunção *EsperaEstimada*(*x*, *30-60*) ∨ *EsperaEstimada*(*x*, >60), poderíamos usar o literal único *EsperaLonga*(*x*). O conjunto de operações de generalização e especialização pode ser facilmente estendido para esse fim.

O algoritmo de espaço de versão puro foi aplicado pela primeira vez no sistema Meta-DENDRAL, projetado com a finalidade de aprender regras para prever como as moléculas se dividiriam em fragmentos em um espectrômetro de massa (Buchanan e Mitchell, 1978). O Meta-DENDRAL foi capaz de gerar regras suficientemente inovadoras para merecer a publicação em um periódico de química analítica — o primeiro conhecimento científico real gerado por um programa de computador. Ele também foi utilizado no elegante sistema LEX (Mitchell *et al*., 1983), que era capaz de aprender a resolver problemas de integração simbólica estudando seus próprios sucessos e fracassos. Embora os métodos de espaço de versão provavelmente não sejam práticos na maioria dos problemas de aprendizagem do mundo real, em grande parte devido ao ruído, eles oferecem uma boa indicação da estrutura lógica do espaço de hipóteses.

19.2 CONHECIMENTO EM APRENDIZAGEM

A seção precedente descreveu a configuração mais simples para aprendizagem indutiva. Para compreender o papel do conhecimento *a priori*, precisamos discutir os relacionamentos lógicos entre hipóteses, descrições de exemplos e classificações. Seja *Descrições* a representação da conjunção de todas as descrições de exemplos no conjunto de treinamento, e seja *Classificações* o valor que denota a conjunção de todas as classificações de exemplos. Então, uma *Hipótese* que "explica as observações" deve satisfazer à seguinte propriedade (lembre-se de que ⊨ significa "consequência lógica"):

$$\mathit{Hipótese} \wedge \mathit{Descrições} \models \text{Classificações}. \tag{19.3}$$

Chamamos essa relação de **restrição de consequência lógica**, na qual *Hipótese* é a "incógnita". A aprendizagem indutiva pura significa resolver essa restrição, em que *Hipótese* é extraída de algum espaço de hipóteses predefinido. Por exemplo, se considerarmos uma árvore de decisão como uma fórmula lógica (veja a Equação 19.1), uma árvore de decisão consistente com todos os exemplos satisfará a Equação 19.3. Se não impusermos *nenhuma* restrição sobre a forma lógica da hipótese, é claro que *Hipótese* = *Classificações* também satisfará a restrição. A navalha de Ockham nos diz que devemos preferir hipóteses *pequenas* e consistentes, e então tentaremos fazer algo melhor do que simplesmente memorizar os exemplos.

Esse quadro simples livre do conhecimento de aprendizagem indutiva persistiu até o início da década de 1980. A abordagem moderna consiste em projetar agentes que *já sabem algo* e estão tentando aprender algo mais. Talvez isso não pareça uma percepção muito profunda, mas faz grande diferença no modo como projetamos agentes. Além disso, ela também poderá ter alguma relevância para nossas teorias sobre o funcionamento da própria ciência. A ideia geral é mostrada esquematicamente na Figura 19.6.

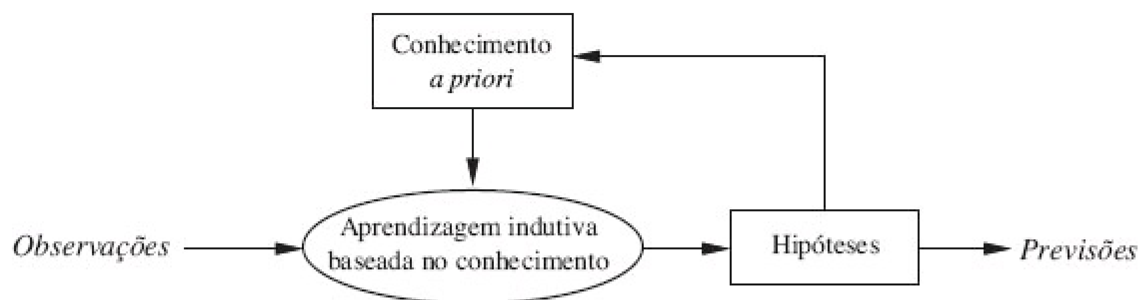


Figura 19.6 Um processo de aprendizagem cumulativa utiliza e amplia seu estoque de conhecimento prático ao longo do tempo.

Se quisermos construir um agente de aprendizagem autônomo que utilize conhecimento prático, devemos de alguma forma obter o conhecimento prático, a fim de usá-lo nos novos episódios de aprendizagem. Esse método tem de ser ele próprio um processo de aprendizagem. A história de vida do agente será então caracterizada por um desenvolvimento *cumulativo* ou *incremental*. Presume-se que o agente possa começar sem nada, realizando induções no vácuo como um pequeno programa de indução pura. Porém, uma vez tendo comido o fruto da árvore do conhecimento, ele não poderá mais procurar tais especulações ingênuas e deverá usar seu conhecimento prático para aprender mais e com maior eficiência. Então, a questão é como realmente fazê-lo.

19.2.1 Alguns exemplos simples

Vamos considerar alguns exemplos de senso comum de aprendizagem com conhecimento prático. Muitos casos aparentemente racionais de comportamento inferencial diante de observações claramente não seguem os princípios simples da indução pura.

- Às vezes, saltamos para conclusões gerais depois de apenas uma observação. Certa vez, Gary Larson desenhou uma caricatura em que um homem das cavernas com óculos, chamado Zog, está

assando seu lagarto na ponta de uma vara. Ele é assistido por uma multidão pasma de contemporâneos menos intelectuais, que estavam usando as mãos nuas para segurar seus alimentos sobre o fogo. Essa esclarecedora experiência foi suficiente para convencer os assistentes de um princípio geral de arte culinária indolor.

- Ou, então, considere o caso do viajante que vai ao Brasil e encontra um brasileiro. Ao ouvi-lo falar português, conclui de imediato que os brasileiros falam português, ainda que, ao descobrir que seu nome é Fernando, ele não conclua que todos os brasileiros se chamem Fernando. Exemplos semelhantes aparecem em ciência. Por exemplo, quando um aluno iniciante em física mede a densidade e a condutância de uma amostra de cobre a uma temperatura específica, ele está bastante confiante na generalização desses valores para todos os pedaços de cobre. Apesar disso, ao medir a massa da amostra, ele nem sequer considera a hipótese de que todos os fragmentos de cobre tenham essa mesma massa. Por outro lado, seria bastante razoável fazer tal generalização em relação a todas as moedas de um centavo.
- Finalmente, considere o caso de um aluno da área médica ignorante em termos farmacológicos, mas sofisticado em termos de diagnóstico, observando uma sessão de consulta entre um paciente e um especialista em infectologia. Depois de uma série de perguntas e respostas, o especialista diz ao paciente que ele deve iniciar o tratamento com um antibiótico específico. O aluno deduz então a regra geral de que esse antibiótico específico é eficaz para determinado tipo de infecção.

☞ Todos esses são casos em que *o uso do conhecimento prático permite aprendizagem muito mais rápida do que se poderia esperar de um programa de indução pura*.

19.2.2 Alguns esquemas gerais

Em cada um dos exemplos precedentes, pode-se apelar para o conhecimento *a priori* com a finalidade de justificar as generalizações escolhidas. Agora, vamos examinar os tipos de restrições de consequência lógica que estão operando em cada caso. As restrições envolverão o conhecimento da *Base*, além da *Hipótese* e das *Descrições* e *Classificações* observadas.

No caso do assado de lagarto, o homem das cavernas generaliza, *explicando* o sucesso da vara usada como espeto: ela suporta o lagarto, ao mesmo tempo em que mantém a mão longe do fogo. A partir dessa explicação, eles podem deduzir uma regra geral: que qualquer objeto longo, rígido e de ponta aguçada pode ser usado para assar pequenos víveres de carne macia. Esse tipo de processo de generalização é chamado **aprendizagem baseada na explanação**, ou **ABE**. Note que a regra geral *decorre logicamente* do conhecimento prático que o homem das cavernas possui. Consequentemente, as restrições de consequência lógica satisfeitas pela ABE são:

$$Hipótese \land Descrições \models Classificações$$
$$Base \models Hipótese.$$

☞ Como a ABE utiliza a Equação 19.3, ela foi considerada inicialmente um modo melhor de aprender a partir de exemplos. Porém, por exigir que o conhecimento prático seja suficiente para

explicar a *Hipótese* que, por sua vez, explica as observações, *na realidade, o agente não aprende nada factualmente novo a partir da instância*. O agente *poderia ter* derivado o exemplo do que já sabia, embora isso talvez tivesse exigido uma quantidade pouco razoável de computação. Agora, a ABE é visualizada como um método para converter teorias de princípios básicos em conhecimento útil, de aplicação especial. Descrevemos algoritmos para ABE na Seção 19.3.

A situação de nosso viajante no Brasil é bem diferente, pois ele não pode explicar necessariamente por que Fernando fala daquele jeito, a menos que conheça as bulas papais. Além disso, a mesma generalização seria proveniente de um viajante completamente ignorante da história colonial. O conhecimento *a priori* relevante nesse caso é que, em qualquer país específico, a maioria das pessoas tende a falar o mesmo idioma; por outro lado, não consideramos que Fernando seja o nome de todos os brasileiros porque essa espécie de regularidade não é válida para nomes. De modo semelhante, o aluno de física iniciante também teria dificuldade para explicar os valores específicos que descobre para a condutância e a densidade do cobre. No entanto, ele sabe que o material do qual um objeto é composto e sua temperatura determinam juntos sua condutância. Em cada caso, o conhecimento *a priori da Base* se refere à **relevância** de um conjunto de características para o predicado objetivo. Esse conhecimento, *juntamente com as observações*, permite ao agente deduzir uma regra nova e geral que explica as observações:

$$\begin{aligned}&Hipótese \wedge Descrições \models Classificações,\\&Base \wedge Descrições \wedge Classificações \models Hipótese.\end{aligned} \qquad (19.4)$$

Chamamos a esse tipo de generalização **aprendizagem baseada na relevância**, ou **ABR** — embora o nome não seja padrão. Note que, enquanto a ABR faz uso do conteúdo das observações, ela não produz hipóteses que vão além do conteúdo lógico do conhecimento prático e das observações. Trata-se de uma forma *dedutiva* de aprendizagem e não pode levar em conta por si só a criação de novo conhecimento a partir do nada.

No caso do aluno de medicina que observa o especialista, supomos que o conhecimento *a priori* do aluno é suficiente para deduzir a doença *D* do paciente a partir dos sintomas. Porém, esse conhecimento não é suficiente para explicar o fato de o médico prescrever um remédio específico *M*. O aluno precisa propor outra regra, ou seja, que *M* geralmente é eficaz contra a doença *D*. Dada essa regra e o conhecimento *a priori* do aluno, este pode agora explicar por que o especialista prescreve *M* nesse caso específico. Podemos generalizar esse exemplo para apresentar a restrição de consequência lógica

$$Base \wedge Hipótese \wedge Descrições \models Classificações. \qquad (19.5)$$

Ou seja, *o conhecimento prático e a nova hipótese se combinam para explicar os exemplos*. Como ocorre no caso da aprendizagem indutiva pura, o algoritmo de aprendizagem deve propor hipóteses tão simples quanto possível e consistentes com essa restrição. Os algoritmos que satisfazem a restrição 19.5 são chamados algoritmos de **aprendizagem indutiva baseada no conhecimento**, ou **AIBC**.

Os algoritmos de AIBC, descritos em detalhes na Seção 19.5, foram estudados principalmente no campo da **programação em lógica indutiva**, ou **PLI**. Em sistemas de PLI, o conhecimento *a priori* desempenha dois papéis fundamentais na redução da complexidade da aprendizagem:

1. Como qualquer hipótese gerada deve ser consistente com o conhecimento *a priori* e também com as novas observações, o tamanho efetivo do espaço de hipóteses é reduzido para incluir apenas aquelas teorias que são consistentes com o que já é conhecido.
2. Para qualquer conjunto de observações dado, o tamanho da hipótese exigida para construir uma explanação relativa às observações pode ser muito reduzido porque o conhecimento *a priori* estará disponível para ajudar as novas regras a explicar as observações. Quanto menor a hipótese, mais fácil será encontrá-la.

Além de permitir o uso do conhecimento *a priori* em indução, os sistemas de PLI podem formular hipóteses em lógica de primeira ordem geral, em vez de formular essas hipóteses na limitada linguagem de atributos do Capítulo 18. Isso significa que eles podem aprender em ambientes que não podem ser entendidos por sistemas mais simples.

19.3 APRENDIZAGEM BASEADA NA EXPLANAÇÃO

Conforme explicamos na introdução a este capítulo, a aprendizagem baseada na explanação é um método para extrair regras gerais de observações individuais. Como exemplo, considere o problema da diferenciação e da simplificação de expressões algébricas (Exercício 9.17). Se diferenciarmos uma expressão como X^2 em relação a X, obteremos $2X$ (note que usamos letra maiúscula para representar a incógnita aritmética X, a fim de distingui-la da variável lógica x). Em um sistema de raciocínio lógico, o objetivo poderia ser expresso como $ASK(Derivada(X^2, X) = d, BC)$, com solução $d = 2X$.

Qualquer pessoa que conheça cálculo diferencial poderá ver essa solução “por inspeção” como resultado da prática na resolução de tais problemas. Um aluno que encontrar tais problemas pela primeira vez ou um programa sem qualquer experiência, terá um trabalho muito mais difícil. A aplicação das regras-padrão de diferenciação eventualmente produz a expressão $1 \times (2 \times (X^{(2-1)}))$, e mais tarde ela é simplificada para $2X$. Na implementação de programação em lógica dos autores, isso requer 136 etapas de prova, das quais 99 são despendidas em ramificações sem saída na prova. Depois de tal experiência, gostaríamos que o programa resolvesse o mesmo problema muito mais rapidamente na próxima vez que ele surgisse.

A técnica de **memoização** tem sido usada há longo tempo em ciência da computação para acelerar programas salvando os resultados da computação. A ideia básica das funções de memo é acumular um banco de dados de pares de entrada/saída; quando a função é chamada, primeiro ela examina o banco de dados para ver se pode evitar resolver o problema desde o início. A aprendizagem baseada na explanação tira bom proveito dessa etapa, criando regras *gerais* que cobrem uma classe inteira de casos. No caso da diferenciação, a memoização lembraria que a derivada de X^2 em relação a X é $2X$, mas deixaria o agente calcular a derivada de Z^2 em relação a Z desde o início. Gostaríamos de ser capazes de extrair a regra geral segundo a qual, para qualquer incógnita aritmética u, a derivada de u^2 em relação a u é $2u$ (pode ser produzida uma regra mesmo mais geral para un, mas o exemplo atual é sufuciente). Em termos lógicos, isso é expresso pela regra

$$IncógnitaAritmética(u) \Rightarrow Derivada(u^2, u) = 2u.$$

Se a base de conhecimento contém tal regra, qualquer caso novo que seja uma instância dessa regra poderá ser resolvido de imediato.

É claro que esse é apenas um exemplo trivial de um fenômeno muito geral. Uma vez que algo é compreendido, pode ser generalizado e reutilizado em outras circunstâncias. O fato se torna uma etapa "óbvia" e pode então ser usado como um bloco de construção na resolução de problemas ainda mais complexos. Alfred North Whitehead (1911), coautor com Bertrand Russell de *Principia Mathematica*, escreveu: "*A civilização avança ampliando o número de operações importantes que podemos executar sem pensar a respeito delas",* talvez aplicando ele mesmo a ABE à sua compreensão de eventos como a descoberta de Zog. Se você compreendeu a ideia básica do exemplo de diferenciação, seu cérebro já está tentando ativamente extrair os princípios gerais da aprendizagem baseada na explanação a partir dele. Note que você não tinha *ainda* criado a ABE antes de ver o exemplo. Como os homens das cavernas observando Zog, você (e nós) precisávamos de um exemplo antes de poder gerar os princípios básicos. Esse é o motivo pelo qual *explicar por que* algo é uma boa ideia é muito mais fácil que apresentar a ideia como passo inicial.

19.3.1 Extraindo regras gerais a partir de exemplos

A ideia básica por trás da ABE é primeiro construir uma explanação da observação usando o conhecimento *a priori* e depois estabelecer uma definição da classe de casos para os quais a mesma estrutura de explanação pode ser usada. Essa definição fornece a base para uma regra que cobre todos os casos na classe. A "explanação" pode ser uma prova lógica, mas, de modo mais geral, pode ser qualquer processo de raciocínio ou de resolução de problemas cujos passos sejam bem definidos. A chave é ser capaz de identificar as condições necessárias para que os mesmos passos se apliquem a outro caso.

Usaremos como nosso sistema de raciocínio o provador simples de teoremas de encadeamento para trás descrito no Capítulo 9. A árvore de prova para $Derivada(X^2, X) = 2X$ é muito grande para ser usada como exemplo e, portanto, usaremos um problema mais simples para ilustrar o método de generalização. Suponha que nosso problema seja simplificar $1 \times (0 + X)$. A base de conhecimento inclui as regras a seguir:

$$
\begin{aligned}
&Reescrever(u, v) \land Simplificar(v, w) \Rightarrow Simplificar(u, w) \,.\\
&Primitiva(u) \Rightarrow Simplificar(u, u) \,.\\
&IncógnitaAritmética(u) \Rightarrow Primitiva(u) \,.\\
&Número(u) \Rightarrow Primitiva(u) \,.\\
&Reescrever(1 \times u, u) \,.\\
&Reescrever(0 + u, u).\\
&\vdots
\end{aligned}
$$

A prova de que a resposta é X é mostrada na metade superior da Figura 19.7. Na realidade, o método de ABE constrói duas árvores de prova simultaneamente. A segunda árvore de prova utiliza um objetivo *variabilizado*, no qual as constantes do objetivo original são substituídas por variáveis. À medida que a prova original prossegue, a prova variabilizada prossegue no mesmo passo, usando *exatamente as mesmas aplicações de regra*. Isso poderia fazer algumas das variáveis se tornarem

instanciadas. Por exemplo, para usar a regra *Reescrever*$(1 \times u, u)$, a variável x no subobjetivo *Reescrever*$(x \times (y + z), v)$ deve estar vinculada a 1. De modo semelhante, y deve estar vinculada a 0 no subobjetivo *Reescrever*$(y + z, v')$, a fim de utilizar a regra *Reescrever*$(0 + u, u)$. Uma vez que temos a árvore de prova generalizada, tomamos as folhas (com as vinculações necessárias) e formamos uma regra geral para o predicado objetivo:

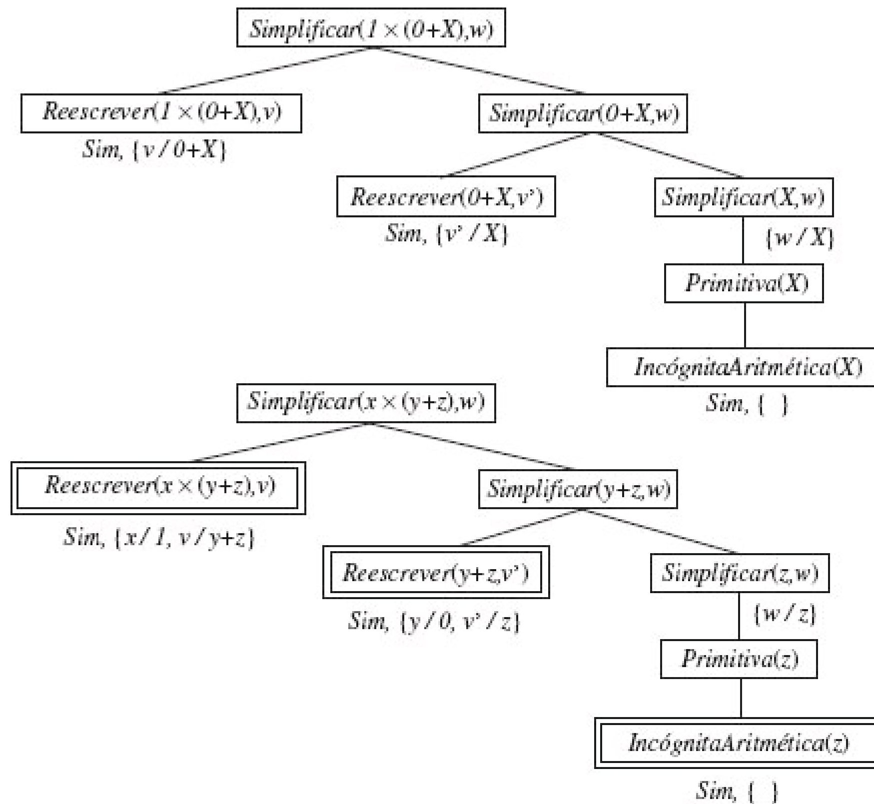


Figura 19.7 Árvores de prova para o problema de simplificação. A primeira árvore mostra a prova para a instância do problema original, da qual podemos derivar:

$$IncógnitaAritmética(z) \Rightarrow Simplificar(1\ \ (0 + z), z)$$

A segunda árvore mostra a prova para uma instância de problema com todas as constantes substituídas por variáveis, da qual podemos derivar uma variedade de outras regras.

$$Reescrever(1 \times (0 + z), 0 + z) \wedge Reescrever(0 + z, z) \wedge IncógnitaAritmética(z) \\ \Rightarrow Simplificar(1 \times (0 + z), z) .$$

Observe que as duas primeiras condições no lado esquerdo são verdadeiras, *não importando o valor de z*. Portanto, podemos descartá-las da regra, formando

$$IncógnitaAritmética(z) \Rightarrow Simplificar(1 \times (0 + z), z) .$$

Em geral, as condições podem ser descartadas da regra final se elas não impõem nenhuma restrição sobre as variáveis do lado direito da regra, porque a regra resultante ainda será verdadeira e será mais eficiente. Note que não podemos descartar a condição *IncógnitaArtimética*(z) porque nem todos os valores possíveis de z são incógnitas aritméticas. Valores que não são incógnitas aritméticas talvez exijam formas diferentes de simplificação: por exemplo, se z fosse 2×3, a simplificação correta de $1 \times (0 + (2 \times 3))$ seria 6 e não 2×3. Para recapitular, o processo básico de ABE funciona assim:

1. Dado um exemplo, construa uma prova de que o predicado objetivo se aplica ao exemplo usando o conhecimento prático disponível.
2. Em paralelo, construa uma árvore de prova generalizada para o objetivo variabilizado, utilizando os mesmos passos de inferência da prova original.
3. Construa uma nova regra cujo lado esquerdo consista nas folhas da árvore de prova e cujo lado direito seja o objetivo variabilizado (depois da aplicação das vinculações necessárias a partir da prova generalizada).
4. Descarte quaisquer condições do lado esquerdo que sejam verdadeiras independentemente dos valores das variáveis no objetivo.

19.3.2 Como melhorar a eficiência

A árvore de prova generalizada da Figura 19.7 realmente gera mais de uma regra generalizada. Por exemplo, se encerrarmos, ou **podarmos**, o crescimento da ramificação da direita na árvore de prova quando ela alcançar o passo *Primitiva*, obteremos a regra:

$$Primitiva(z) \Rightarrow Simplificar(1 \times (0 + z), z) .$$

Embora *mais geral*, essa regra é tão válida quanto a regra que utiliza *IncógnitaAritmética*, porque cobre os casos em que z é um número. Podemos extrair uma regra ainda mais geral efetuando a poda depois do passo *Simplificar*($y + z$, w) gerando a regra:

$$Simplificar(y + z, w) \Rightarrow Simplificar(1 \times (y + z), w) .$$

Em geral, uma regra pode ser extraída de *qualquer subárvore parcial* da árvore de prova generalizada. Agora, temos um problema: qual dessas regras escolheremos?

A escolha de qual regra gerar se reduz à questão da eficiência. Existem três fatores envolvidos na análise de ganhos de eficiência a partir de ABE:

1. A adição de grande número de regras pode diminuir a velocidade do processo de raciocínio porque o mecanismo de inferência ainda tem de verificar essas regras, mesmo em casos nos quais elas não produzem uma solução. Em outras palavras, ela aumenta o **fator de ramificação** no espaço de busca.
2. Para compensar a redução da velocidade de raciocínio, as regras derivadas devem oferecer aumentos significativos em velocidade para os casos que elas abrangem. Esses aumentos surgem principalmente porque as regras derivadas evitam becos sem saída que de outro forma seriam seguidos, mas também porque elas encurtam a prova propriamente dita.
3. As regras derivadas devem ser tão gerais quanto possível, de forma que elas se apliquem ao maior conjunto possível de casos.

Uma abordagem comum para assegurar que as regras derivadas são eficientes é insistir na **operacionalidade** de cada subobjetivo na regra. Um subobjetivo é operacional se é “fácil” resolvê-lo. Por exemplo, é fácil resolver o subobjetivo *Primitiva*(z), pois isso exige no máximo dois passos,

enquanto o subobjetivo *Simplificar*$(y + z, w)$ poderia levar a uma quantidade arbitrária de inferência, dependendo dos valores de y e z. Se um teste para operacionalidade for executado em cada passo na construção da prova generalizada, poderemos podar o restante de uma ramificação assim que um subobjetivo operacional for encontrado, mantendo apenas o subobjetivo operacional como um elemento da conjunção da nova regra.

Infelizmente, em geral existe um compromisso entre operacionalidade e generalidade. Os subobjetivos mais específicos costumam ser mais fáceis de resolver, mas abrangem um número menor de casos. Além disso, a operacionalidade é uma questão de grau: definitivamente, a existência de um ou dois passos é operacional, mas o que dizer de 10 ou 100?

Por fim, o custo de resolver determinado subobjetivo depende de quais outras regras estarão disponíveis na base de conhecimento. Ela pode aumentar ou diminuir à medida que mais regras são adicionadas. Desse modo, os sistemas de ABE na realidade enfrentam um problema de otimização muito complexo na tentativa de maximizar a eficiência de dada base de conhecimento inicial. Algumas vezes é possível derivar um modelo matemático do efeito sobre a eficiência global da adição de determinada regra e empregar esse modelo para selecionar a melhor regra a adicionar. Entretanto, a análise poderá se tornar muito complicada, especialmente quando regras recursivas estiverem envolvidas. Uma abordagem promissora é atacar o problema da eficiência de modo empírico, simplesmente adicionando diversas regras e verificando quais delas são úteis e de fato aceleram o processo.

A análise empírica da eficiência está realmente no núcleo da ABE. O que temos chamado livremente de "eficiência de determinada base de conhecimento" é de fato a complexidade do caso médio em uma distribuição de problemas. *Por generalização a partir de exemplos de problemas passados*, a *ABE torna a base de conhecimento mais eficiente para o tipo de problemas que é razoável esperar*. Isso funciona desde que a distribuição de exemplos passados seja aproximadamente a mesma dos exemplos futuros — a mesma suposição usada para a aprendizagem PAC na Seção 18.5. Se o sistema de ABE for cuidadosamente elaborado, será possível obter acelerações significativas. Por exemplo, um sistema de linguagem natural muito grande baseado em Prolog projetado para tradução vocal entre os idiomas sueco e inglês foi capaz de alcançar desempenho de tempo real somente pela aplicação de ABE ao processo de análise (Samuelsson e Rayner, 1991).

19.4 APRENDIZAGEM COM O USO DE INFORMAÇÕES DE RELEVÂNCIA

Nosso viajante no Brasil parece ser capaz de fazer uma generalização confiante em relação ao idioma falado por outros brasileiros. A inferência é sancionada por seu conhecimento prático, isto é, de que as pessoas de dado país (em geral) falam o mesmo idioma. Podemos expressar essa inferência em lógica de primeira ordem como a seguir:[2]

$$Nacionalidade(x, n) \land Nacionalidade(y, n) \land Idioma(x, l) \Rightarrow Idioma(y, l). \quad (19.6)$$

(Tradução literal: "Se x e y têm a mesma nacionalidade n e x fala o idioma l, então y também fala

esse idioma.”) Não é difícil demonstrar que, a partir dessa sentença e da observação de que

$$Nacionalidade(Fernando, Brasil) \land Idioma(Fernando, Português),$$

a conclusão a seguir é consequência lógica (veja o Exercício 19.1):

$$Nacionalidade(x, Brasil) \Rightarrow Idioma(x, Português).$$

Sentenças como 19.6 expressam uma forma estrita de relevância: dada a nacionalidade, o idioma é completamente determinado (em outras palavras: o idioma é uma função da nacionalidade). Essas sentenças são chamadas **dependências funcionais** ou **determinações**. Elas ocorrem de forma tão comum em certos tipos de aplicações (por exemplo, na definição de projetos de bancos de dados) que uma sintaxe especial é usada para escrevê-las. Adotaremos a notação de Davies (1985):

$$Nacionalidade(x, n) \succ Idioma(x, l).$$

Como usual, isso é simplesmente um ajuste sintático, mas torna claro que a determinação é na realidade um relacionamento entre os predicados: a nacionalidade determina o idioma. As propriedades relevantes que determinam a condutância e a densidade podem ser expressas de modo semelhante:

$$Material(x, m) \land Temperatura(x, t) \succ Condutância(x, \rho);$$
$$Material(x, m) \land Temperatura\ (x, t) \succ Densidade\ (x, d).$$

As generalizações correspondentes decorrem logicamente das determinações e observações.

19.4.1 Determinação do espaço de hipóteses

Embora as determinações sancionem conclusões gerais relativas a todos os brasileiros ou a todos os pedaços de cobre em determinada temperatura, é claro que elas não podem produzir uma teoria preditiva geral para *todas* as nacionalidades ou para *todas* as temperaturas e materiais, a partir de um exemplo único. Seu principal efeito pode ser visto como a limitação do espaço de hipóteses que o agente de aprendizagem precisa considerar.

☞ Por exemplo, na previsão da condutância, é necessário considerar apenas o material e a temperatura, podendo-se ignorar a massa, a propriedade, o dia da semana, o presidente atual, e assim por diante. As hipóteses podem, sem dúvida, incluir termos que, por sua vez, são determinados pelo material e pela temperatura, como a estrutura molecular, a energia térmica ou a densidade de elétrons livres. *As determinações especificam um vocabulário de base suficiente a partir do qual devem ser construídas hipóteses relativas ao predicado-alvo*. Essa declaração pode ser comprovada mostrando-se que dada determinação é logicamente equivalente a uma declaração de que a definição correta do predicado-alvo é um elemento do conjunto de todas as definições que podem ser expressas com a utilização dos predicados do lado esquerdo da determinação.

Intuitivamente, é claro que uma redução no tamanho do espaço de hipóteses deve tornar mais fácil aprender o predicado-alvo. Usando os resultados básicos da teoria de aprendizagem computacional

(Seção 18.5), podemos quantificar os ganhos possíveis. Primeiro, lembre-se de que, para funções booleanas, $\log(|\mathcal{H}|)$ exemplos têm de convergir para uma hipótese razoável, onde $|\mathcal{H}|$ é o tamanho do espaço de hipóteses. Se o aluno tem n características booleanas com que construir hipóteses, então, na ausência de restrições adicionais, $|\mathcal{H}| = O(2^{2^n})$ e, assim, o número de exemplos será $O(2^n)$. Se a determinação contém d predicados no lado esquerdo, o aluno exigirá apenas $O(2^d)$ exemplos, uma redução de $O(2^{n-d})$.

19.4.2 Aprendizagem e utilização de informações de relevância

Conforme declaramos na introdução a este capítulo, o conhecimento *a priori* é útil em aprendizagem, mas ele próprio também precisa ser aprendido. Para fornecer um roteiro completo de aprendizagem baseado em relevância, devemos portanto fornecer um algoritmo de aprendizagem para determinações. O algoritmo de aprendizagem que apresentaremos agora se baseia em uma tentativa direta de encontrar a determinação mais simples consistente com as observações. Uma determinação $P \succ Q$ nos diz que, se quaisquer exemplos coincidirem em P, eles também deverão coincidir em Q. Então, uma determinação é consistente com um conjunto de exemplos se todo par que corresponde nos predicados do lado esquerdo também corresponde no predicado-alvo, isto é, tem a mesma classificação. Por exemplo, suponha que tenhamos os exemplos de medições de condutância em amostras dos materiais dados a seguir:

Amostra	Massa	Temperatura	Material	Tamanho	Condutância
S1	12	26	Cobre	3	0,59
S1	12	100	Cobre	3	0,57
S2	24	26	Cobre	6	0,59
S3	12	26	Chumbo	2	0,05
S3	12	100	Chumbo	2	0,04
S4	24	26	Chumbo	4	0,05

A determinação consistente mínima é $Material \land Temperatura \succ Condutância$. Existe uma determinação não mínima, mas consistente, isto é, $Massa \land Tamanho \land Temperatura \succ Condutância$. Isso é consistente com os exemplos porque massa e tamanho determinam densidade e, em nosso conjunto de dados, não temos dois materiais diferentes com a mesma densidade. Como sempre, precisaríamos de um conjunto de amostras maior, a fim de eliminar uma hipótese quase correta.

Existem vários algoritmos possíveis para encontrar determinações consistentes mínimas. A abordagem mais óbvia é conduzir uma busca pelo espaço de determinações verificando todas as determinações com um predicado, dois predicados, e assim por diante, até encontrar uma determinação consistente. Vamos supor uma representação baseada em atributo simples, como a que usamos para aprendizagem de árvores de decisão no Capítulo 18. Uma determinação d será representada pelo conjunto de atributos no lado esquerdo porque o predicado-alvo é considerado

fixo. O algoritmo básico está representado na Figura 19.8.

```
função DET-CONSISTENTE-MÍNIMA(E, A) retorna um conjunto de atributos
  entradas: E, um conjunto de exemplos
            A, um conjunto de atributos de tamanho n
  para i = 0 a n faça
    para cada subconjunto A de A de tamanho i faça
      se DET-CONSISTENTE?(Ai, E) então retornar Ai
```

```
função DET-CONSISTENTE?(A, E) retorna um valor-verdade
  entradas: A, um conjunto de atributos
            E, um conjunto de exemplos
  variáveis locais: H, uma tabela de hash
  para cada exemplo e em E faça
    se algum exemplo em H tem os mesmos valores que e para os atributos A
      mas tem uma classificação diferente então retornar falso
    armazenar a classe de e em H, indexada pelos valores correspondentes aos atributos A do exen
  retornar verdadeiro
```

Figura 19.8 Algoritmo para encontrar uma determinação consistente mínima.

A complexidade de tempo desse algoritmo depende do tamanho da menor determinação consistente. Suponha que essa determinação tenha *p* atributos além do número total *n* de atributos. Então, o algoritmo não a encontrará até pesquisar os subconjuntos de *A* de tamanho *p*. Existem $\binom{n}{p} = O(n^p)$ desses subconjuntos; consequentemente, o algoritmo é exponencial no tamanho da determinação mínima. Na verdade, o problema é NP-completo e, assim, não podemos esperar um resultado melhor no caso geral. No entanto, na maioria dos domínios, haverá uma estrutura local suficiente (veja no Capítulo 14 uma definição de domínios localmente estruturados) para que *p* seja pequeno.

Dado um algoritmo para aprender determinações, um agente de aprendizagem tem uma alternativa para construir uma hipótese mínima, dentro da qual poderá aprender o predicado-alvo. Por exemplo, podemos combinar DET-CONSISTENTE-MÍNIMA com o algoritmo APRENDIZAGEMEM-ÁRVORE-DE-DECISÃO. Isso vai gerar um algoritmo de aprendizagem baseado em árvore de decisão denominado AADBR (ou aprendizagem em árvore de decisão baseada em relevância) que primeiro identifica umconjunto mínimo de atributos relevantes e depois repassa esse conjunto ao algoritmo de árvore de decisão para aprendizagem. Diferentemente de APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO, o algoritmo AADBR aprende e utiliza simultaneamente informações de relevância, com o objetivo de minimizar seu espaço de hipóteses. Esperamos que AADBR aprenda com maior rapidez que APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO e, de fato, é isso o que acontece. A Figura 19.9 mostra o desempenho de aprendizagem para os dois algoritmos sobre dados gerados aleatoriamente para uma função que depende de apenas 5 de 16 atributos. É óbvio que, em casos em que todos os atributos disponíveis forem relevantes, o AADBR não mostrará nenhuma

vantagem.

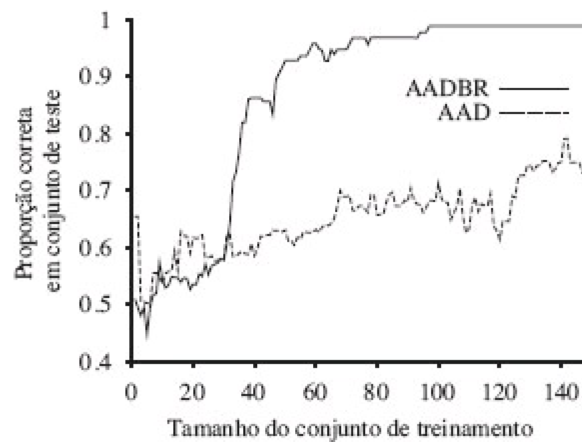


Figura 19.9 Comparação de desempenho entre AADBR e APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO sobre dados gerados aleatoriamente para uma função-alvo que depende de apenas 5 entre 16 atributos.

Esta seção apenas arranhou a superfície do campo de **tendência declarativa**, que tem por objetivo compreender como o conhecimento *a priori* pode ser usado para identificar o espaço de hipóteses apropriado, dentro do qual será realizada a busca da definição-alvo correta. Existem muitas perguntas sem resposta:

- Como os algoritmos podem ser estendidos para tratar ruído?
- Podemos manipular variáveis de valores contínuos?
- Como outros tipos de conhecimento *a priori* podem ser empregados, além das determinações?
- De que maneira os algoritmos podem ser generalizados para cobrir qualquer teoria de primeira ordem, em lugar de abordarem apenas uma representação baseada em atributos?

Algumas dessas perguntas serão tratadas na próxima seção.

19.5 PROGRAMAÇÃO EM LÓGICA INDUTIVA

A programação em lógica indutiva (PLI) combina métodos indutivos com o poder das representações de primeira ordem, concentrando-se em particular na representação de teorias como programas lógicos.[3] Ela ganhou popularidade por três razões. Primeiro, a PLI oferece uma abordagem rigorosa para o problema de aprendizagem indutiva baseada no conhecimento geral. Em segundo lugar, ela oferece algoritmos completos para induzir teorias gerais de primeira ordem a partir de exemplos que podem, portanto, aprender com sucesso em domínios nos quais os algoritmos baseados em atributos dificilmente se aplicam. Um exemplo está na aprendizagem de como as estruturas de proteínas se entrelaçam (Figura 19.10). A configuração tridimensional de uma molécula de proteína não pode ser representada de modo razoável por um conjunto de atributos porque a configuração se refere inerentemente a *relacionamentos* entre objetos, e não a atributos de um único objeto. A lógica de primeira ordem é uma linguagem apropriada para descrever os relacionamentos. Em terceiro lugar, a programação em lógica indutiva produz hipóteses que são (relativamente) fáceis

de ler para os seres humanos. Por exemplo, a tradução em linguagem comum da Figura 19.10 pode ser averiguada e criticada por biólogos. Isso significa que os sistemas de programação em lógica indutiva podem participar do ciclo científico de experimentação, geração de hipóteses, debate e refutação. Tal participação não seria possível para sistemas que geram classificadores do tipo “caixa-preta”, como redes neurais.

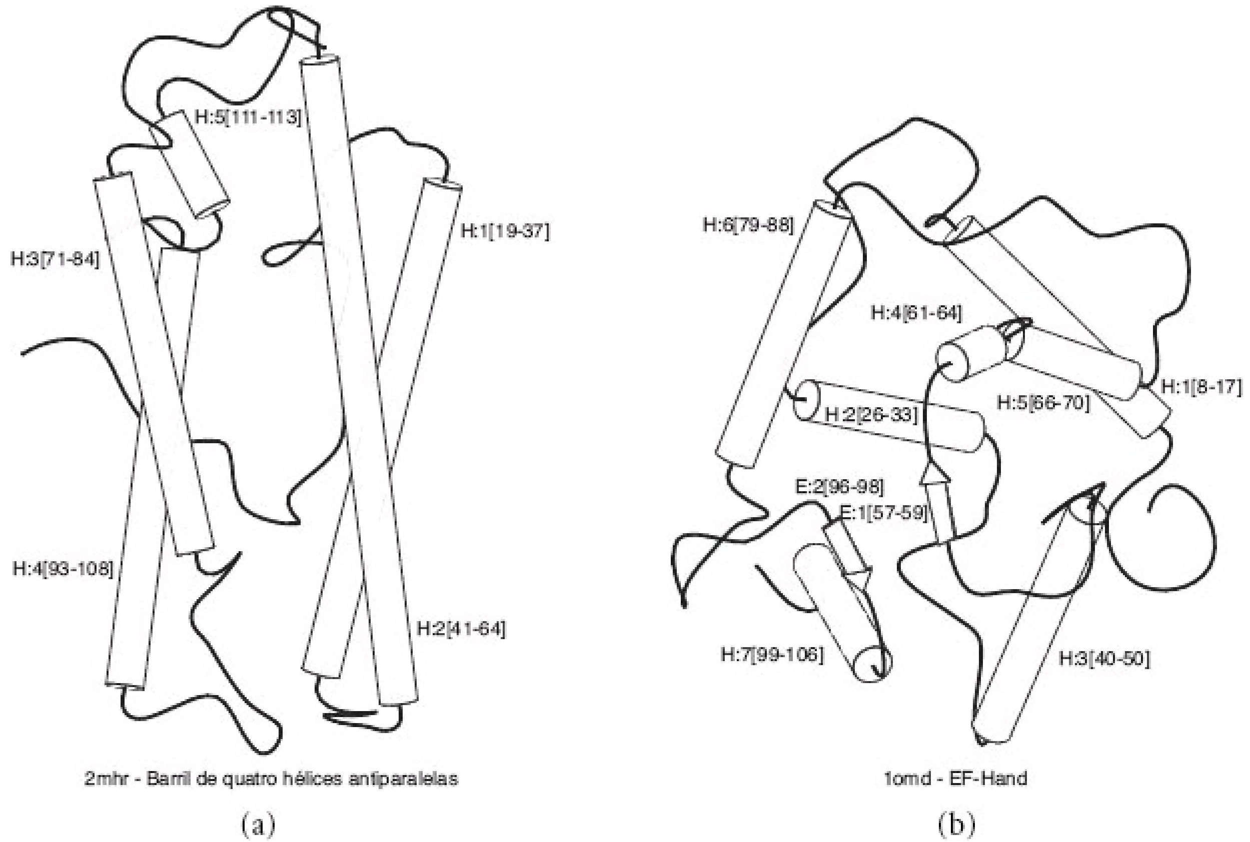


Figura 19.10 (a) e (b) mostram exemplos positivos e negativos, respectivamente, do conceito de “barril de quatro hélices antiparalelas” no domínio de entrelaçamento de proteínas. Cada exemplo de estrutura é codificado em uma expressão lógica de cerca de 100 conjunções, como *ComprimentoTotal*(*D*2*mhr*, 118) ∧ *NúmeroHélices*(*D*2*mhr*, 6) ∧…. A partir dessas descrições e de classificações como *Entrelaçar*(BARRIL-DE-QUATRO-HELICES-ANTIPARALELAS, *D*2*mhr*), o sistema PLI PROGOL (Muggleton, 1995) aprendeu a regra a seguir:

$$\begin{array}{l} Entrelaçar(\text{BARRIL-DE-QUATRO-HELICES-ANTIPARALELAS}, p) \Leftarrow \\ Hélice(p, h_1) \wedge Comprimento(h_1, \text{ALTO}) \wedge Posição(p, h_1, n) \\ \wedge\ (1 \leq n \leq 3) \wedge Adjacente(p, h_1, h_2) \wedge Hélice(p, h_2). \end{array}$$

Essa espécie de regra não poderia ser aprendida, ou mesmo representada, por um mecanismo baseado em atributos, como os que vimos em capítulos anteriores. A regra pode ser traduzida para linguagem comum como: “A proteína *p* tem classe de entrelaçamento de ‘barril de quatro hélices antiparalelas’ se contém uma longa hélice h_1 em uma posição de estrutura secundária entre 1 e 3, e h_1 está ao lado de uma segunda hélice”.

19.5.1 Um exemplo

Vimos na Equação 19.5 que a solução do problema geral de indução baseada no conhecimento consiste em “resolver” a restrição de consequência lógica

$$Base \wedge Hipótese \wedge Descrições \models Classificações$$

para a incógnita *Hipótese*, dado o conhecimento de *Base* e exemplos descritos por *Descrições* e *Classificações*. Para ilustrar esse fato, utilizaremos o problema da aprendizagem de relacionamentos de família a partir de exemplos. As descrições consistirão em uma árvore genealógica estendida, descrita em termos das relações *Mãe*, *Pai* e *Casado* e das propriedades *Homem* e *Mulher*. Como exemplo, usaremos a árvore genealógica do Exercício 8.14 mostrada na Figura 19.11. As descrições correspondentes são:

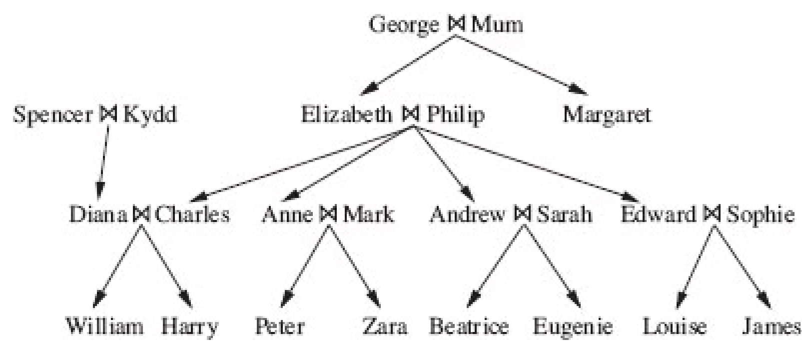


Figura 19.11 Árvore genealógica típica.

Pai(Philip, Charles)	*Pai(Philip, Anne)*	…
Mãe(Mum, Margaret)	*Mãe(Mum, Elizabeth)*	…
Casado(Diana, Charles)	*Casado(Elizabeth, Philip)*	…
Homem(Philip)	*Homem(Charles)*	…
Mulher(Beatrice)	*Mulher(Margaret)*	…

As sentenças em *Classificações* dependem do conceito-alvo que está sendo aprendido. Por exemplo, talvez queiramos aprender *Avô*, *Cunhado* ou *Ancestral*. Para *Avô*, o conjunto completo de *Classificações* contém 20 × 20 = 400 conjuntos da forma:

Avô(Mum, Charles)	*Avô(Elizabeth, Beatrice)*	…
¬Avô(Mum, Harry)	*¬Avô(Spencer, Peter)*	…

É claro que poderíamos aprender a partir de um subconjunto desse conjunto completo.

O objetivo de um programa de aprendizagem indutivo é apresentar um conjunto de sentenças para a *Hipótese* tal que a restrição de consequência lógica seja satisfeita. Suponha, por um instante, que o agente não tenha nenhum conhecimento prático: a *Base* está vazia. Então, uma solução possível para *Hipótese* é:

$$\begin{aligned} Avô(x, y) \Leftrightarrow & [\exists z\, Mãe(x, z) \wedge Mãe(z, y)] \\ \vee\ & [\exists z\, Mãe(x, z) \wedge Pai\,(z, y)] \\ \vee\ & [\exists z\, Pai\,(x, z) \wedge Mãe(z, y)] \\ \vee\ & [\exists z\, Pai\,(x, z) \wedge Pai(z, y)] \end{aligned}$$

Note que um algoritmo de aprendizagem baseado em atributos, como APRENDIZAGEM-EMÁRVORE-DE-DECISÃO, não chegará a lugar nenhum na resolução desse problema. Para

expressar *Avô* como um atributo (isto é, um predicado unário), precisaríamos transformar *pares* de pessoas em objetos:

Avô(⟨*Mum*, *Charles*⟩)…

Assim, ficamos paralisados, tentando representar as descrições do exemplo. Os únicos atributos possíveis são itens horríveis como:

PrimeiroElementoÉMãeDeElizabeth(⟨*Mum*, *Charles*⟩).

A definição de *Avô* em termos desses atributos simplesmente se torna uma grande disjunção de casos específicos que não se generaliza de modo algum para novos exemplos. *Os algoritmos de aprendizagem baseados em atributos são incapazes de aprender predicados relacionais*. Desse modo, uma das principais vantagens dos algoritmos de PLI é sua aplicabilidade a uma faixa muito mais ampla de problemas, incluindo problemas relacionais.

O leitor certamente terá notado que um pouco de conhecimento prático ajudaria na representação da definição de *Avô*. Por exemplo, se a *Base* incluísse a sentença

$$Pai(x,y) \Leftrightarrow [Mãe(x,y) \vee Pai(x,y)],$$

a definição de *Avô* seria reduzida a

$$Avô(x,y) \Leftrightarrow [\exists z\ Pai^{*}(x,z) \wedge Pai^{*}(z,y)].$$

Isso mostra como o conhecimento prático pode reduzir drasticamente o tamanho de hipóteses exigidas para explicar as observações.

Também é possível para os algoritmos de PLI *criar* novos predicados, a fim de facilitar a expressão de hipóteses explicativas. Considerando-se os dados de exemplo mostrados antes, é completamente razoável para o programa de PLI propor um predicado adicional, que chamaríamos de "*Pai*", a fim de simplificar as definições dos predicados-alvo. Os algoritmos que podem gerar novos predicados são chamados algoritmos de **indução construtiva**. É claro que a indução construtiva é uma parte necessária do quadro de aprendizagem cumulativo esboçado na introdução. Ela ainda é um dos problemas mais difíceis de aprendizagem de máquina, mas algumas técnicas de PLI fornecem mecanismos efetivos para alcançá-la.

No restante deste capítulo, estudaremos as duas abordagens principais para PLI. A primeira utiliza uma generalização de métodos de árvores de decisão, e a segunda emprega técnicas baseadas na inversão de uma prova de resolução.

19.5.2 Métodos top-down para aprendizagem indutiva

A primeira abordagem para PLI funciona a partir de uma regra muito geral e segue especializando-a gradualmente de modo que ela se adapte aos dados. Isso é essencialmente o que acontece na aprendizagem de árvores de decisão, na qual uma árvore de decisão é gradualmente ampliada até se

tornar consistente com as observações. Para realizar a PLI, utilizamos literais de primeira ordem em lugar de atributos, e a hipótese é um conjunto de cláusulas em vez de uma árvore de decisão. Esta seção descreve o FOIL (Quinlan, 1990), um dos primeiros programas de PLI.

Vamos supor que estamos tentando aprender uma definição do predicado *Avô*(x, y), usando os mesmos dados de família de antes. Como ocorre com a aprendizagem de árvores de decisão, podemos dividir os exemplos em exemplos positivos e negativos. Os exemplos positivos são

$$\langle George, Anne\rangle, \langle Philip, Peter\rangle, \langle Spencer, Harry\rangle, \ldots$$

e os exemplos negativos são

$$\langle George, Elizabeth\rangle, \langle Harry, Zara\rangle, \langle Charles, Philip\rangle, \ldots$$

Note que cada exemplo é um *par* de objetos porque *Avô* é um predicado binário. Ao todo, existem 12 exemplos positivos na árvore genealógica e 388 exemplos negativos (todos os outros pares de pessoas).

O FOIL constrói um conjunto de cláusulas, cada uma com *Avô*(x, y) como o início. As cláusulas devem classificar os 12 exemplos positivos como instâncias do relacionamento *Avô*(x, y), ao mesmo tempo que eliminam os 388 exemplos negativos. As cláusulas são cláusulas de Horn, com a extensão que literais negados são permitidos no corpo de uma cláusula e interpretados usando negação por falha, como em Prolog. A cláusula inicial tem um corpo vazio:

$$\Rightarrow Avô(x, y).$$

Essa cláusula classifica todo exemplo como positivo e, assim, precisa ser especializada. Fazemos isso adicionando literais um a um no lado esquerdo. Aqui estão três inclusões potenciais:

$$Pai(x, y) \Rightarrow Avô(x, y).$$
$$Pai(x, z) \Rightarrow Avô(x, y).$$
$$Pai(x, z) \Rightarrow Avô(x, y).$$

(Note que estamos supondo que uma cláusula que define *Pai* já faz parte do conhecimento prático.) A primeira dessas três cláusulas classifica incorretamente todos os 12 exemplos positivos como negativos e, portanto, pode ser ignorada. A segunda e a terceira cláusulas concordam com todos os exemplos positivos, mas a segunda é incorreta sobre uma fração maior dos exemplos negativos — o dobro de vezes porque permite mães, bem como pais. Consequentemente, preferimos a terceira cláusula.

Agora, precisamos especializar ainda mais essa cláusula, a fim de eliminar os casos em que x é o pai de algum z, mas z não é um pai de y. A inclusão do único literal *Pai*(z, y) fornece

$$Pai(x, z) \land Pai(z, y) \Rightarrow Avô(x, y),$$

que classifica corretamente todos os exemplos. O FOIL descobrirá e escolherá esse literal, resolvendo assim a tarefa de aprendizagem. Em geral, a solução é um conjunto de cláusulas de Horn, cada uma das quais implica o predicado-alvo. Por exemplo, se não tivéssemos o predicado *Pai* em nosso vocabulário, a solução poderia ser

$$Pai(x, z) \wedge Pai(z, y) \Rightarrow Avô(x, y)$$
$$Pai(x, z) \wedge Mãe(z, y) \Rightarrow Avô(x, y).$$

Observe que cada uma dessas cláusulas abrange alguns dos exemplos positivos que, juntos, abrangem todos os exemplos positivos, e que a CLÁUSULA-NOVA é projetada de tal forma que nenhuma cláusula vai abranger incorretamente um exemplo negativo. Em geral, o FOIL terá de buscar por muitas cláusulas malsucedidas antes de encontrar uma solução correta.

Esse exemplo é uma ilustraçãomuito simples de como o FOIL opera. Um esboço do algoritmo completo é mostrado na Figura 19.12. Em essência, o algoritmo constrói repetidamente uma cláusula, literal por literal, até que ela concorde com algum subconjunto dos exemplos positivos e com nenhum dos exemplos negativos. Então, os exemplos positivos cobertos pela cláusula são removidos do conjunto de treinamento, e o processo continua até não restar nenhum exemplo positivo. As duas principais sub-rotinas que serão explicadas são NOVOS-LITERAIS, a qual constrói todos os novos literais possíveis para adicionar à cláusula, e ESCOLHER-LITERAL, que seleciona um literal para inclusão.

função FOIL(*exemplos*, *destino*) **retorna** um conjunto de cláusulas de Horn
  entradas: *exemplos*, conjunto de exemplos
      destino, um literal correspondente ao predicado objetivo
  variáveis locais: *cláusulas*, um conjunto de cláusulas, inicialmente vazio
  enquanto *exemplos* contém exemplos positivos **faça**
    cláusula ← NOVA-CLÁUSULA(*exemplos*, *destino*)
    remover exemplos cobertos por *cláusula* de *exemplos*
    adicionar *cláusula* a *cláusulas*
  retornar *cláusulas*

---

função NOVA-CLÁUSULA(*exemplos*, *destino*) **retorna** uma cláusula de Horn
  variáveis locais: *cláusula*, uma cláusula com *destino* como início e um corpo vazio
      l, um literal a ser adicionado à cláusula
      exemplos_estendidos, um conjunto de exemplos com valores para novas var
  exemplos_estendidos ← *exemplos*
  enquanto *exemplos_estendidos* contém exemplos negativos **faça**
    l ← ESCOLHER-LITERAL(NOVOS-LITERAIS(*cláusula*), *exemplos_estendidos*)
    acrescentar *l* ao corpo de *cláusula*
    exemplos_estendidos ← conjunto de exemplos criados pela aplicação de ESTENDER-EXEM
      em *exemplos_estendidos*
  retornar *cláusula*

---

função ESTENDER-EXEMPLO(*exemplo*, *literal*) **retorna**
  se *exemplo* satisfaz o *literal*
    então retornar o conjunto de exemplos criados estendendo-se *exemplo* com cada valor con
variável em *literal*

senão retornar o conjunto vazio

Figura 19.12 Um esboço do algoritmo FOIL para aprendizagem de conjuntos de cláusulas de Horn de primeira ordem a partir de exemplos.

NOVOS-LITERAIS recebe uma cláusula e constrói todos os possíveis literais "úteis" que poderiam ser adicionados à cláusula. Vamos usar como exemplo a cláusula

$$Pai(x, z) \Rightarrow Avô(x, y).$$

Existem três tipos de literais que podem ser adicionados:

1. *Literais que utilizam predicados:* os literais podem ser negados ou não negados, podendo ser utilizado qualquer predicado existente (inclusive o predicado objetivo), e todos os argumentos devem ser variáveis. Qualquer variável pode ser usada para qualquer argumento do predicado, com uma única restrição: cada literal deve incluir *pelo menos uma* variável de um literal anterior ou do início da cláusula. Literais como $Mãe(z, u)$, $Casado(z, z)$, $\neg Homem(y)$ e $Avô(v, x)$ são permitidos, enquanto $Casado(u, v)$ não é. Note que o uso do predicado do início da cláusula permite ao FOIL aprender definições *recursivas*.
2. *Literais de igualdade e desigualdade*: esses literais relacionam variáveis que já aparecem na cláusula. Por exemplo, poderíamos adicionar $z \neq x$. Esses literais também podem incluir constantes especificadas pelo usuário. Para aprendizagem aritmética, poderíamos utilizar 0 e 1 e, para aprendizagem de funções de listas, poderíamos empregar a lista vazia [ ].
3. *Comparações aritméticas*: quando lidamos com funções de variáveis contínuas, literais como $x > y$ e $y \leq z$ podem ser adicionados. Como na aprendizagem em árvores de decisão, um valor de limiar constante pode ser escolhido para maximizar a capacidade de distinção do teste.

O fator de ramificação resultante nesse espaço de busca é muito grande (veja o Exercício 19.6), mas o FOIL também pode usar informações de tipo para reduzi-lo. Por exemplo, se o domínio incluísse números, NOVOS-LITERAIS e ESCOLHER-LITERAL são explicados no texto. bem como pessoas, as restrições de tipo impediriam NOVOS-LITERAIS de gerar literais como $Pai(x, n)$, onde x é uma pessoa e n é um número.

ESCOLHER-LITERAL utiliza uma heurística de certa forma semelhante ao ganho de informações para decidir que literal adicionar. Os detalhes exatos não são tão importantes aqui, e diversas variações foram experimentadas. Uma característica adicional interessante do FOIL é o uso da navalha de Ockham para eliminar algumas hipóteses. Se uma cláusula se tornar mais longa (de acordo com alguma métrica) que o comprimento total dos exemplos positivos que a cláusula explica, essa cláusula não será considerada uma hipótese potencial. Essa técnica fornece um caminho para se evitar cláusulas supercomplexas que inserem ruído nos dados.

O FOIL e seus semelhantes foram usados para aprender uma ampla variedade de definições. Uma das demonstrações mais impressionantes (Quinlan e Cameron-Jones, 1993) envolvia a resolução de uma longa sequência de exercícios sobre funções de processamento de listas do livro didático sobre Prolog de Bratko (1986). Em cada caso, o programa foi capaz de aprender uma definição correta da

função a partir de um conjunto pequeno de exemplos usando as funções anteriormente aprendidas como conhecimento prático.

19.5.3 Aprendizagem indutiva com dedução inversa

A egunda abordagem importante para PLI envolve a inversão do processo normal de prova dedutiva. A **resolução inversa** se baseia na observação de que, se o exemplo *Classificações* decorre de *ConhecimentoPrático* ∧ *Hipótese* ∧ *Descrições*, devemos ser capazes de provar esse fato por resolução (porque a resolução é completa). Se pudermos "executar a prova no sentido inverso", será possível encontrar uma *Hipótese* tal que a prova seja válida. Portanto, a chave é descobrir um modo de inverter o processo de resolução.

Mostraremos um processo de prova em sentido contrário para resolução inversa que consiste em etapas individuais em sentido oposto. Lembre-se de que um passo de resolução comum recebe duas cláusulas C_1 e C_2, e as resolve para produzir o **resolvente** C. Um passo de resolução inversa recebe um resolvente C e produz duas cláusulas C_1 e C_2, tais que C seja o resultado da resolução de C_1 e C_2. Outra alternativa seria receber um resolvente C e uma cláusula C_1, e produzir uma cláusula C_2 tal que C seja o resultado da resolução de C_1 e C_2.

As primeiras etapas em um processo de resolução inversa são mostradas na Figura 19.13, onde nos concentramos no exemplo positivo *Avô*(*George*, *Anne*). O processo começa no final da prova (mostrado na parte inferior da figura). Tomamos o resolvente C como a cláusula vazia (isto é, uma contradição) e C_2 como ¬*Avô*(*George*, *Anne*), que é a negação do exemplo de objetivo. O primeiro passo inverso recebe C e C_2, e gera a cláusula *Avô*(*George*, *Anne*) para C_1. O passo seguinte recebe essa cláusula como C e a cláusula *Pai*(*Elizabeth*, *Anne*) como C_2, e gera a cláusula

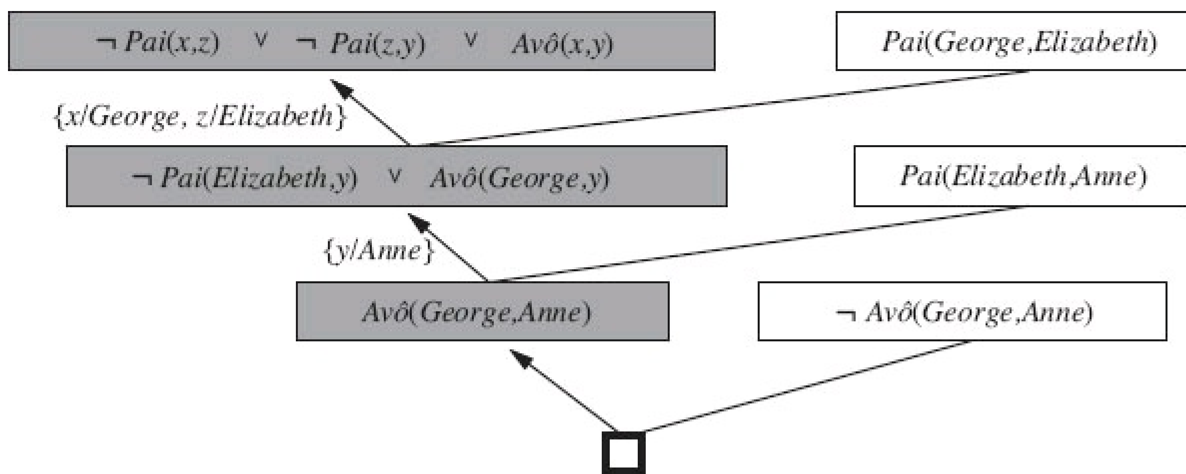


Figura 19.13 Passos iniciais em um processo de resolução inversa. As cláusulas sombreadas são geradas por passos de resolução inversa a partir da cláusula à direita e da cláusula abaixo delas. As cláusulas não sombreadas vêm de *Descrições* e *Classificações* (incluindo *Classificações* negadas).

$$\neg Pai(Elizabeth, y) \lor Avô(George, y)$$

como C_1. A etapa final trata essa cláusula como o resolvente. Tendo *Pai*(*George*, *Elizabeth*) como

C_2, uma cláusula C_1 possível é a hipótese:

$$Pai(x, z) \land Pai(z, y) \Rightarrow Pai(x, y).$$

Agora, temos uma prova de resolução de que a hipótese, as descrições e o conhecimento prático impõem a classificação *Avô*(*George*, *Anne*).

É claro que a resolução inversa envolve uma busca. Cada etapa de resolução inversa é não determinística porque, para qualquer C, pode haver muitas ou até mesmo um número infinito de cláusulas C_1 e C_2 que se resolvem em C. Por exemplo, em vez de escolher $\neg Pai(Elizabeth, y) \lor Avô(George, y)$ para C_1 na última etapa da Figura 19.13, a etapa de resolução inversa poderia ter escolhido qualquer das sentenças a seguir:

$$\neg Pai(Elizabeth, Anne) \lor Avô(George, Anne).$$
$$\neg Pai(z, Anne) \lor Avô(George, Anne).$$
$$\neg Pai(z, y) \lor Avô(George, y).$$
$$\vdots$$

(Veja os Exercícios 19.4 e 19.5.) Além disso, as cláusulas que participam de cada passo podem ser escolhidas a partir do conhecimento de *Base*, a partir do exemplo *Descrições*, a partir da negação de *Classificações* ou a partir de cláusulas hipotéticas que já tenham sido geradas na árvore de resolução inversa. O grande número de possibilidades indica um grande fator de ramificação (e, portanto, uma busca ineficiente) sem controles adicionais. Várias abordagens para ajustar a busca foram experimentadas em sistemas de PLI implementados:

1. Escolhas redundantes podem ser eliminadas — por exemplo, gerando apenas as hipóteses mais específicas possíveis e exigindo que todas as cláusulas hipotéticas sejam consistentes umas com as outras e com as observações. Este último critério elimina a cláusula $\neg Pai(z, y) \lor Avô(George, y)$, listada antes.
2. A estratégia de prova pode ser restrita. Por exemplo, vimos no Capítulo 9 que a **resolução linear** é uma estratégia restrita completa. A resolução linear produz árvores de prova que têm uma estrutura de ramificação linear — a árvore inteira segue uma linha, com apenas uma única ramificação de cláusulas fora de linha (como na Figura 19.13)
3. A linguagem de representação pode ser restrita, por exemplo, eliminando símbolos de funções ou permitindo apenas cláusulas de Horn. Para exemplificar, o PROGOL opera com cláusulas de Horn usando a **consequência lógica inversa**. A ideia é alterar a restrição de consequência lógica

$$Base \land Hipótese \land Descrições \models Classificações$$

para a forma logicamente equivalente

$$Base \land Descrições \land \neg Classificações \models \neg Hipótese.$$

A partir disso, podemos utilizar um processo semelhante ao da dedução normal de cláusulas de Horn de Prolog, com negação por falha, para derivar *Hipótese*. Pelo fato de ser restrito a

cláusulas de Horn, esse é um método incompleto, mas pode ser mais eficiente que a resolução completa. Também é possível aplicar a inferência completa com a consequência lógica inversa (Inoue, 2001).

4. A inferência pode ser feita com verificação de modelos, em prova de teoremas. O sistema PROGOL (Muggleton, 1995) utiliza uma forma de verificação de modelos para limitar a busca. Isto é, de modo semelhante à programação de conjuntos de resposta, ele gera valores possíveis para variáveis lógicas e verifica a consistência.
5. A inferência pode ser realizada com cláusulas proposicionais básicas em lugar da lógica de primeira ordem. O sistema LINUS (Lavrauc e Duzeroski, 1994) funciona convertendo teorias de primeira ordem em lógica proposicional, resolvendo-as com um sistema de aprendizagem proposicional e depois fazendo a conversão de volta. O trabalho com fórmulas proposicionais pode ser mais eficiente em alguns problemas, como vimos no caso de SATPLAN no Capítulo 10.

19.5.4 Fazendo descobertas com programação em lógica indutiva

Um procedimento de resolução inversa que inverte uma estratégia de resolução completa é, em princípio,um algoritmo completo para aprendizagem de teorias de primeira ordem. Ou seja, se alguma incógnita *Hipótese* gerar um conjunto de exemplos, um procedimento de resolução inversa poderá gerar *Hipótese* a partir dos exemplos. Essa observação sugere uma possibilidade interessante: suponha que os exemplos disponíveis incluam uma variedade de trajetórias de corpos em queda livre. Um programa de resolução inversa seria teoricamente capaz de deduzir a lei da gravidade? Sem dúvida, a resposta é sim porque a lei da gravidade permite explicar os exemplos, dados os cálculos matemáticos práticos adequados. De modo semelhante, poderíamos imaginar que o eletromagnetismo, a mecânica quântica e a teoria da relatividade também estejam dentro do escopo de programas de PLI. É claro que eles também estão dentro do escopo de um macaco com uma máquina de escrever; ainda precisaremos de heurísticas melhores e de novas alternativas para estruturar o espaço de busca.

Uma coisa que os sistemas de resolução inversa *farão* por você será criar novos predicados. Com frequência, essa habilidade é vista como algo mágico porque os computadores muitas vezes são considerados máquinas que “meramente funcionam com as informações que recebem”. De fato, novos predicados recaem diretamente fora da etapa de resolução inversa. O caso mais simples surge quando consideramos por hipótese duas novas cláusulas, C_1 e C_2, dada uma cláusula C. A resolução de C_1 e C_2 elimina um literal que as duas cláusulas compartilham; consequentemente, é bastante possível que o literal eliminado contivesse um predicado que não aparece em C. Desse modo, ao se trabalhar no sentido contrário, uma possibilidade é gerar um novo predicado a partir do qual será reconstruído o literal que falta.

A Figura 19.14 mostra um exemplo em que o novo predicado P é gerado no processo de aprendizagem de uma definição para *Ancestral*. Uma vez gerado, P pode ser usado em etapas posteriores de resolução inversa. Por exemplo, uma etapa posterior poderia definir hipoteticamente $Mãe(x, y) \Rightarrow P(x, y)$. Desse modo, o novo predicado P tem seu significado limitado pela geração de

hipóteses que o envolvem. Outro exemplo poderia levar à restrição $Pai(x, y) \Rightarrow P(x, y)$. Em outras palavras, o predicado P é o que normalmente imaginamos como o relacionamento de *Pai*. Como mencionamos antes, a criação de novos predicados pode reduzir de forma significativa o tamanho da definição do predicado objetivo. Consequentemente, pela inclusão da habilidade para criar novos predicados, os sistemas de resolução inversa com frequência podem resolver problemas de aprendizagem que são inviáveis com outras técnicas.

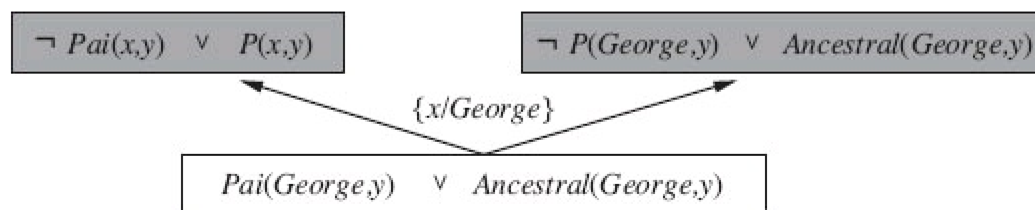


Figura 19.14 Uma etapa de resolução inversa que gera um novo predicado P.

Algumas das mais profundas revoluções na ciência vêm da criação de novos predicados e funções — por exemplo, a descoberta da aceleração por Galileu ou a descoberta da energia térmica por Joule. Uma vez que esses termos estão disponíveis, a descoberta de novas leis se torna (relativamente) fácil. A parte difícil reside na percepção de que alguma nova entidade, mantendo um relacionamento específico com entidades existentes, permitirá que todo um conjunto de observações seja explicado com uma teoria muito mais simples e mais elegante do que tudo o que existia anteriormente.

Até agora, os sistemas de PLI não fizeram descobertas no nível de Galileu ou Joule, mas suas descobertas foram consideradas publicáveis na literatura científica. Por exemplo, no *Journal of Molecular Biology*, Turcotte *et al*. (2001) descrevem a descoberta automatizada de regras para entrelaçamento de proteínas pelo programa de PLI PROGOL. Muitas das regras descobertas pelo programa PROGOL poderiam ter sido derivadas a partir de princípios conhecidos, mas a maioria delas não tinha sido publicada anteriormente como parte de um banco de dados biológico padrão (veja um exemplo na Figura 19.10). Em um trabalho relacionado, Srinivasan *et al*. (1994) lidaram com o problema de descobrir regras baseadas na estrutura molecular para o caráter mutagênico de compostos nitrocompostos aromáticos. Esses compostos são encontrados em gases emitidos por automóveis. No caso de 80% dos compostos encontrados em um banco de dados padrão, é possível identificar quatro características importantes, e a regressão linear sobre essas características supera a PLI. Para os 20% restantes, as características sozinhas não podem ser previstas, e a PLI identifica relacionamentos que lhe permitem superar o desempenho da regressão linear, das redes neurais e das árvores de decisão. O mais impressionante é que King *et al*. (2009) dotaram um robô com a capacidade de realizar experimentos de biologia molecular e estenderam as técnicas de PLI para incluir um projeto experimental, criando assim um cientista autônomo que realmente descobriu novos conhecimentos sobre a genômica funcional da levedura.

Para todos esses exemplos, parece que a habilidade de representar relações e de usar o conhecimento prático contribui para o alto desempenho da PLI. O fato de as regras encontradas pela PLI poderem ser interpretadas pelos seres humanos contribui para a aceitação dessas técnicas em periódicos de biologia, e não apenas em periódicos de ciência da computação.

A PLI realizou contribuições para outras ciências além da biologia. Uma das mais importantes é o

processamento de linguagem natural, no qual a PLI foi usada para extrair informações relacionais complexas a partir de texto. Esses resultados estão resumidos no Capítulo 23.

19.6 RESUMO

Este capítulo investigou vários modos pelos quais o conhecimento *a priori* pode ajudar um agente a aprender a partir de novas experiências. Como muito conhecimento *a priori* é expresso em termos de modelos relacionais, e não em modelos baseados em atributos, também estudamos sistemas que permitem a aprendizagem de modelos relacionais. Os pontos importantes são:

- O uso do conhecimento *a priori* na aprendizagem leva a um quadro de **aprendizagem cumulativa**, no qual os agentes de aprendizagem melhoram sua habilidade de aprender à medida que adquirem mais conhecimento.
- O conhecimento *a priori* ajuda na aprendizagem, eliminando hipóteses que de outra forma seriam consistentes e “completando” a explicação de exemplos, permitindo assim hipóteses mais curtas. Com frequência, essas contribuições resultam em aprendizagem mais rápida a partir de um número menor de exemplos.
- Compreender os diferentes papéis lógicos assumidos pelo conhecimento *a priori*, expressos por **restrições de consequência lógica**, ajuda a definir uma variedade de técnicas de aprendizagem.
- A **aprendizagem baseada em explanação** (ABE) extrai regras gerais a partir de exemplos isolados *explicando* os exemplos e generalizando a explanação. Ela fornece um método dedutivo que transforma o conhecimento de princípios básicos em experiência útil, eficiente e de uso especial.
- A **aprendizagem baseada na relevância** (ABR) utiliza o conhecimento *a priori* na forma de determinações para identificar os atributos relevantes, gerando assim um espaço de hipóteses reduzido e acelerando a aprendizagem. A ABR também permite generalizações dedutivas a partir de exemplos isolados.
- A **aprendizagem indutiva baseada no conhecimento** (AIBC) encontra hipóteses indutivas que explicam conjuntos de observações com a ajuda de conhecimento prático.
- As técnicas de **programação em lógica indutiva** (PLI) executam a AIBC sobre conhecimento expresso em lógica de primeira ordem. Os métodos de PLI podem aprender conhecimento relacional que não pode ser expresso em sistemas baseados em atributos.
- A PLI pode ser realizada com uma abordagem top-down de refinar uma regra muito geral ou por meio de uma abordagem bottom-up de inversão do processo dedutivo.
- Os métodos de PLI geram naturalmente novos predicados com os quais podem ser expressas novas teorias concisas e se mostram promissores como sistemas de formação de teorias científicas de uso geral.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Embora o uso do conhecimento *a priori* na aprendizagem pudesse parecer um tópico natural para

os filósofos da ciência, pouco trabalho formal foi realizado até recentemente. A obra *Fact, Fiction, and Forecast*, do filósofo Nelson Goodman (1954), refutou a suposição anterior de que a indução era simplesmente uma questão de ver exemplos suficientes de alguma proposição universalmente quantificada e depois adotá-la como hipótese. Considere, por exemplo, a hipótese "todas as esmeraldas são *grue*" (*green* + *blue* : verde + azul), onde *grue* significa "verdes, se observadas antes do tempo *t*, mas azuis se observadas depois disso". Em qualquer instante até *t*, poderíamos ter observado milhões de instâncias confirmando a regra de que as esmeraldas são *grue* e nenhuma instância contrária a essa regra, e ainda assim estaríamos pouco dispostos a adotá-la. Isso pode ser explicado apenas pelo apelo à função do conhecimento *a priori* relevante no processo de indução. Goodman propõe uma variedade de tipos diferentes de conhecimento *a priori* que poderiam ser úteis, inclusive uma versão de determinações chamadas **super-hipóteses**. Infelizmente, as ideias de Goodman nunca foram adotadas na aprendizagem de máquina.

A abordagem da **melhor-hipótese-atual** é uma ideia antiga em filosofia (Mill, 1843). Um trabalho antigo em psicologia cognitiva também sugeriu que é uma forma natural do conceito de aprendizagem em seres humanos (Bruner *et al*., 1957). Em IA, a abordagem é mais associada com o trabalho de Patrick Winston, cuja tese de Ph.D. (Winston, 1970) abordou o problema de descrição de aprendizagem de objetos complexos. O método de **espaço de versão** (Mitchell, 1977, 1982) toma uma abordagem diferente, mantendo o conjunto de *todas* as hipóteses consistentes e eliminando aquelas consideradas incompatíveis com novos exemplos. A abordagem foi utilizada no sistema especialista Meta-DENDRAL para a química (Buchanan e Mitchell, 1978) e, mais tarde, no sistema LEX de Mitchell (1983), que aprende a resolver problemas de cálculo. O trabalho de Michalski e colegas formou um terceiro segmento influente na série de algoritmos AQ, aprendendo conjuntos de regras lógicas (Michalski, 1969; Michalski *et al*., 1986).

A ABE tem suas raízes nas técnicas usadas pelo planejador STRIPS (Fikes *et al*., 1972). Quando um plano era construído, uma versão generalizada desse plano era guardada em uma biblioteca de planos e usada no planejamento posterior como um **operador macro**. Ideias semelhantes apareceram na arquitetura ACT* de Anderson, sob o título **compilação do conhecimento** (Anderson, 1983), e na arquitetura SOAR, como **formação de bloco** (Laird *et al*., 1986). A **aquisição de esquemas** (DeJong, 1981), a **generalização analítica** (Mitchell, 1982) e a **generalização baseada em restrições** (Minton, 1984) foram precursoras imediatas do rápido crescimento do interesse em ABE, estimulado pelos ensaios de Mitchell *et al*. (1986) e DeJong e Mooney (1986). Hirsh (1987) introduziu o algoritmo de ABE descrito no texto, mostrando como ele podia ser incorporado diretamente a um sistema de programação lógica. Van Harmelen e Bundy (1988) explicam a ABE como uma variante do método de **avaliação parcial** utilizada em sistemas de análise de programas (Jones *et al*., 1993).

O entusiasmo inicial pelo ABE foi temperado pelos achados de Minton (1988) que demonstraram que, sem trabalho extra extensivo, o ABE poderia tornar um programa significativamente lento. A análise probabilística formal da recompensa esperada do ABE pode ser encontrada em Greiner (1989) e Subramanian e Feldman (1990). Em Dietterich (1990) temos uma revisão excelente dos trabalhos iniciais sobre ABE.

Em vez de usar exemplos como foco para generalização, podemos usá-los diretamente para resolver novos problemas, em um processo conhecido como **raciocínio analógico**. Essa forma de

raciocínio varia desde uma forma de raciocínio plausível baseado em grau de semelhança (Gentner, 1983), passando por uma forma de inferência dedutiva baseada em determinações mas exigindo a participação do exemplo (Davies e Russell, 1987), até uma forma de ABE “adiado” que adapta a direção da generalização do exemplo antigo às necessidades do novo problema. Esta última forma de raciocínio analógico é encontrada mais comumente no **raciocínio baseado em casos** (Kolodner, 1993) e na **analogia derivacional** (Veloso e Carbonell, 1993).

As informações de relevância na forma de dependências funcionais foram desenvolvidas primeiro na comunidade de banco de dados, onde foram empregadas para estruturar grandes conjuntos de atributos em subconjuntos manejáveis. As dependências funcionais foram usadas para raciocínio analógico por Carbonell e Collins (1973), e Davies e Russell (Davies, 1985; Davies e Russell, 1987) redescobriram e apresentaram uma análise lógica completa. Seu papel como conhecimento prévio em aprendizagem indutiva foi explorado por Russell e Grosof (1987). A equivalência de determinações a um espaço de hipóteses de vocabulário restrito foi provada em Russell (1988). Os algoritmos de aprendizagem para determinação e o desempenho otimizado obtido pelo AADBR foram mostrados primeiro no algoritmo FOCUS em Almuallim e Dietterich (1991). Tadepalli (1993) descreve um algoritmo engenhoso para aprendizagem com determinações que mostram grandes melhorias na velocidade de aprendizagem.

A ideia de que a aprendizagem indutiva pode ser realizada por dedução inversa remonta a W. S. Jevons (1874), que escreveu: “O estudo da lógica formal e da teoria das probabilidades me levou a adotar a opinião de que não existe algo que se possa considerar um método distinto de indução em contraste com a dedução, mas essa indução é simplesmente um emprego inverso da dedução.” As investigações computacionais começaram com a importante tese de doutorado de Gordon Plotkin (1971) em Edinburgh. Embora Plotkin desenvolvesse muitos dos teoremas e métodos que estão em uso corrente na PLI, ele foi desencorajado por alguns resultados de indecidibilidade para certos subproblemas em indução. O MIS (Shapiro, 1981) reintroduziu o problema de aprender programas de lógica, mas foi visto principalmente como uma contribuição para a teoria de depuração automatizada. O trabalho em indução de regras, como os sistemas ID3 (Quinlan, 1986) e CN2 (Clark e Niblett, 1989), levou ao FOIL (Quinlan, 1990), que, pela primeira vez, permitia a indução prática de regras relacionais. O campo de aprendizagem relacional foi revigorado por Muggleton e Buntine (1988), cujo programa CIGOL incorporou uma versão ligeiramente incompleta da resolução inversa e foi capaz de gerar novos predicados. O método de resolução inversa também aparece em Russel (1986) com um algoritmo simples mostrado em nota de rodapé. O segundo sistema mais importante foi o GOLEM (Muggleton e Feng, 1990), que utiliza um algoritmo de cobertura baseado no conceito de Plotkin de generalização relativa. O ITOU (Rouveirol e Puget, 1989) e o CLINT (De Raedt, 1992) foram outros sistemas dessa época. Mais recentemente, o PROGOL (Muggleton, 1995) adotou uma abordagem híbrida (top-down e bottom-up) para consequência lógica inversa, e foi aplicado a uma série de problemas práticos, especificamente na biologia e em processamento de linguagens naturais. Muggleton (2000) descreve uma extensão do PROGOL para manipular a incerteza na forma de programas lógicos estocásticos.

Uma análise formal de métodos de PLI aparece em Muggleton (1991), em uma grande coletânea de artigos em Muggleton (1992), e em uma coletânea de técnicas e aplicações no livro de Lavrac e Djzzeroski (1994). Page e Srinivasan (2002) fornecem uma visão geral mais recente da história e dos

desafios do campo para o futuro. Os primeiros resultados de complexidade de Haussler (1989) sugeriram que a aprendizagem de sentenças de primeira ordem era intratável. Porém, com melhor compreensão da importância de vários tipos de restrições sintáticas sobre cláusulas, foram obtidos resultados positivos mesmo para cláusulas com recursão (Djzzeroski *et al*., 1992). Os resultados de aprendizagem para PLI foram pesquisados por Kietz e Djzzeroski (1994) e por Cohen e Page (1995).

Embora a PLI agora pareça ser a abordagem dominante para indução construtiva, ela não foi a única abordagem adotada. Os chamados **sistemas de descoberta** se destinam a modelar o processo de descoberta científica de novos conceitos, em geral por uma busca direta no espaço de definições de conceitos. O Automated Mathematician de Doug Lenat, ou AM (Davis e Lenat, 1982), usava heurísticas de descoberta expressas como regras de sistemas especialistas para orientar sua busca de conceitos e conjeturas em teoria elementar dos números. Diferentemente da maioria dos sistemas projetados para raciocínio matemático, o AM não tinha um conceito de prova e só podia fazer conjeturas. Ele redescobriu a conjetura de Goldbach e o teorema de fatoração exclusiva de números primos. A arquitetura do AM foi generalizada no sistema EURISKO (Lenat, 1983) pela adição de um mecanismo capaz de reescrever as heurísticas de descoberta do próprio sistema. O EURISKO foi aplicado em diversas áreas além da descoberta matemática, embora com menos sucesso que o AM. A metodologia do AM e do EURISKO tem gerado controvérsias (Ritchie e Hanna, 1984; Lenat e Brown, 1984).

Outra classe de sistemas de descoberta se destina a operar com dados científicos reais para descobrir novas leis. Os sistemas DALTON, GLAUBER e STAHL (Langley *et al*., 1987) são sistemas baseados em regras que procuram relacionamentos quantitativos em dados experimentais de sistemas físicos; em cada caso, o sistema foi capaz de recapitular uma descoberta conhecida a partir da história da ciência. Os sistemas de descoberta baseados em técnicas probabilísticas — em especial os algoritmos de formação de agrupamentos que descobrem novas categorias — são discutidos no Capítulo 20.

EXERCÍCIOS

19.1 Mostre, por conversão em forma normal conjuntiva e aplicação da resolução, que a conclusão da página 728 referente aos brasileiros é consistente.

19.2 Para cada uma das determinações a seguir, escreva a representação lógica e explique por que a determinação é verdadeira (se for o caso):

- **a.** O desenho e o valor determinam a massa de uma moeda.
- **b.** Para dado programa, a entrada determina a saída.
- **c.** O clima, a ingestão de alimentos, o exercício e o metabolismo determinam o ganho e a perda de peso.
- **d.** A calvície é determinada pela calvície (ou pela falta dela) do avô materno de um indivíduo.

19.3 Uma versão probabilística de determinações seria útil? Sugira uma definição.

19.4 Preencha os valores que faltam para completar as cláusulas C_1 ou C_2 (ou ambas) nos conjuntos

de cláusulas a seguir, dado que C é o resolvente de C_1 e de C_2:

a. $C = Verdadeiro \Rightarrow P(A, B)$, $C_1 = P(x, y) \Rightarrow Q(x, y)$, $C_2 = ??$.

b. $C = Verdadeiro \Rightarrow P(A, B)$, $C_1 =??$, $C_2 =??$.

c. $C = P(x, y) \Rightarrow P(x, f(y))$, $C_1 =??$, $C_2 =??$.

Se existir mais de uma solução possível, forneça um exemplo de cada tipo diferente.

19.5 Suponha que se escreva um programa em lógica que executa um passo de inferência de resolução. Isto é, considere $Resolver(c_1, c_2, c)$ bem-sucedida se c é o resultado da resolução de c_1 e c_2. Normalmente, *Resolver* seria usada como parte de um provador de teoremas, chamando-a com c_1 e c_2 instanciadas para cláusulas específicas, gerando assim o resolvente c. Agora suponha que, em vez disso, ela fosse chamada com c instanciada e com c_1 e c_2 não instanciadas. Essa rotina terá sucesso na geração dos resultados apropriados de uma etapa de resolução inversa? Você precisaria realizar algumas modificações especiais no sistema de programação em lógica para isso funcionar?

19.6 Suponha que o FOIL esteja considerando a possibilidade de adicionar um literal a uma cláusula com a utilização de um predicado binário P e que literais anteriores (inclusive o início da cláusula) contenham cinco variáveis diferentes.

a. Quantos literais funcionalmente diferentes podem ser gerados? Dois literais são funcionalmente idênticos se diferem apenas pelos nomes das *novas* variáveis que contêm.

b. Você poderia encontrar uma fórmula geral para o número de literais diferentes com um predicado de aridade r quando existem n variáveis anteriormente usadas?

c. Por que o FOIL não permite literais que não contêm nenhuma variável usada anteriormente?

19.7 Usando os dados da árvore genealógica da Figura 19.11 ou de um subconjunto dessa árvore, aplique o algoritmo FOIL para aprender uma definição relativa ao predicado *Ancestral*.

---

[1] Os termos "falso positivo" e "falso negativo" são usados em medicina para descrever resultados errôneos de exames de laboratório. O resultado é um falso positivo se indica que o paciente possui a doença quando, na verdade, ele não possui.

[2] Supomos, por questão de simplicidade, que uma pessoa fala apenas um idioma. É claro que a regra também teria de ser aperfeiçoada para países como a Suíça e a Índia.

[3] Talvez fosse apropriado nesse momento para o leitor reportar-se ao Capítulo 7 com a finalidade de examinar alguns conceitos subjacentes, inclusive cláusulas de Horn, forma normal conjuntiva, unificação e resolução.

CAPÍTULO 20

Aprendizagem de modelos probabilísticos

Em que visualizamos a aprendizagem como uma forma de raciocínio com incerteza a partir de observações.

O Capítulo 13 assinalou a prevalência da incerteza em ambientes reais. Os agentes podem manipular a incerteza usando os métodos de probabilidade e teoria da decisão, mas primeiro eles devem aprender suas teorias probabilísticas do mundo a partir da experiência. Este capítulo explica como é possível fazê-lo, pela formulação da tarefa de aprendizagem em si como um processo de inferência probabilística (Seção 20.1). Veremos que uma visão bayesiana da aprendizagem é extremamente poderosa, fornecendo soluções gerais para os problemas de ruído, superadaptação e previsão ótima. Ele também leva em conta o fato de que um agente que não seja onisciente nunca poderá ter certeza sobre qual teoria do mundo é correta, mesmo que ainda tome decisões usando alguma teoria de mundo.

Descreveremos métodos para modelos probabilísticos com aprendizagem — principalmente redes bayesianas — nas Seções 20.2 e 20.3. Uma parte do material deste capítulo é bastante matemática, embora as lições gerais possam ser entendidas sem a necessidade de mergulhar nos detalhes. O leitor pode tirar proveito de revisar o material dos Capítulos 13 e 14 e o Apêndice A.

20.1 APRENDIZAGEM ESTATÍSTICA

Os conceitos fundamentais deste capítulo, da mesma maneira que os do Capítulo 18, são **dados** e **hipóteses**. Aqui, os dados são **evidências**, isto é, instanciações de algumas ou de todas as variáveis aleatórias que descrevem o domínio. Neste capítulo, as hipóteses são teorias probabilísticas de como o domínio funciona, incluindo teorias lógicas como caso especial.

Vamos considerar um exemplo simples. Nosso doce surpresa favorito tem dois sabores: cereja (hum) e lima (eca). O fabricante de doces tem um senso de humor peculiar e embrulha cada pedaço de doce na mesma embalagem opaca, independentemente do sabor. O doce é vendido em sacos muito grandes, dos quais existem cinco tipos conhecidos — novamente, indistinguíveis a partir do exterior:

h_1: 100% cereja

h_2: 75% cereja + 25% lima

h_3: 50% cereja + 50% lima
h_4: 25% cereja + 75% lima
h_5: 100% lima

Dado um novo saco de doces, a variável aleatória H (de *hipótese*) denota o tipo do saco, com valores possíveis h_1 até h_5. É evidente que H não é diretamente observável. À medida que os doces são abertos e inspecionados, são revelados os dados — $D_1, D_2, \ldots, D_n$, onde cada D_i é uma variável aleatória com valores possíveis *cereja* e *lima*. A tarefa básica enfrentada pelo agente é prever o sabor do próximo doce.[1] Apesar de sua trivialidade aparente, esse cenário serve para introduzir muitas questões importantes. O agente realmente precisa deduzir uma teoria de seu mundo, embora seja um mundo muito simples.

A **aprendizagem bayesiana** simplesmente calcula a probabilidade de cada hipótese, considerando-se os dados, e faz previsões de acordo com ela. Isto é, as previsões são feitas com o uso de *todas* as hipóteses, ponderadas por suas probabilidades, em vez de utilizar apenas uma única "melhor" hipótese. Desse modo, a aprendizagem é reduzida à inferência probabilística. Seja $\mathbf{D}$ a representação de todos os dados, com valor observado $\mathbf{d}$; então, a probabilidade de cada hipótese é obtida pela regra de Bayes:

$$P(h_i \mid \mathbf{d}) = \alpha P(\mathbf{d} \mid h_i) P(h_i) . \quad (20.1)$$

Agora, vamos supor que queremos fazer uma previsão sobre uma quantidade desconhecida X. Então, temos:

$$\mathbf{P}(X \mid \mathbf{d}) = \sum_i \mathbf{P}(X \mid \mathbf{d}, h_i) \mathbf{P}(h_i \mid \mathbf{d}) = \sum_i \mathbf{P}(X \mid h_i) P(h_i \mid \mathbf{d}) , \quad (20.2)$$

onde pressupomos que cada hipótese determina uma distribuição de probabilidade sobre X. Essa equação mostra que as previsões são médias ponderadas sobre as previsões das hipóteses individuais. As hipóteses propriamente ditas são em essência "intermediários" entre os dados brutos e as previsões. As quantidades fundamentais na abordagem de Bayes são a **hipótese *a priori***, $P(h_i)$ e a **probabilidade** dos dados sob cada hipótese, $P(\mathbf{d}|h_i)$.

Para nosso exemplo dos doces, vamos supor por enquanto que a distribuição *a priori* sobre $h_1, \ldots, h_5$ seja dada por $\langle 0{,}1,\ 0{,}2,\ 0{,}4,\ 0{,}2,\ 0{,}1 \rangle$, como anunciado pelo fabricante. A probabilidade dos dados é calculada sob a suposição de que as observações são **i.i.d.**, de forma que:

$$P(\mathbf{d} \mid h_i) = \prod_j P(d_j \mid h_i) . \quad (20.3)$$

Por exemplo, suponha que o saco seja realmente um saco só com doces de lima (h_5) e que os primeiros 10 doces sejam todos de lima; então, $P(\mathbf{d} \mid h_3)$ é $0{,}5^{10}$, porque metade dos doces em um saco do tipo h_3 é de doces de lima.[2] A Figura 20.1(a) mostra como as probabilidades posteriores das cinco hipóteses mudam à medida que a sequência de 10 doces de lima é observada. Note que as

probabilidades começam com seus valores *a priori* e, portanto, h_3 é inicialmente a escolha mais provável e permanece assim depois que um doce de lima é desembrulhado. Depois de serem desembrulhados dois doces de lima, h_4 é mais provável; após três ou mais, h_5 (o temido saco só com doces de lima) é o mais provável. Depois de 10 doces seguidos, estamos bastante certos de nosso destino. A Figura 20.1(b) mostra a probabilidade prevista de que o próximo doce seja de lima, com base na Equação 20.2. Como seria de esperar, ela aumenta monotonicamente em direção a 1.

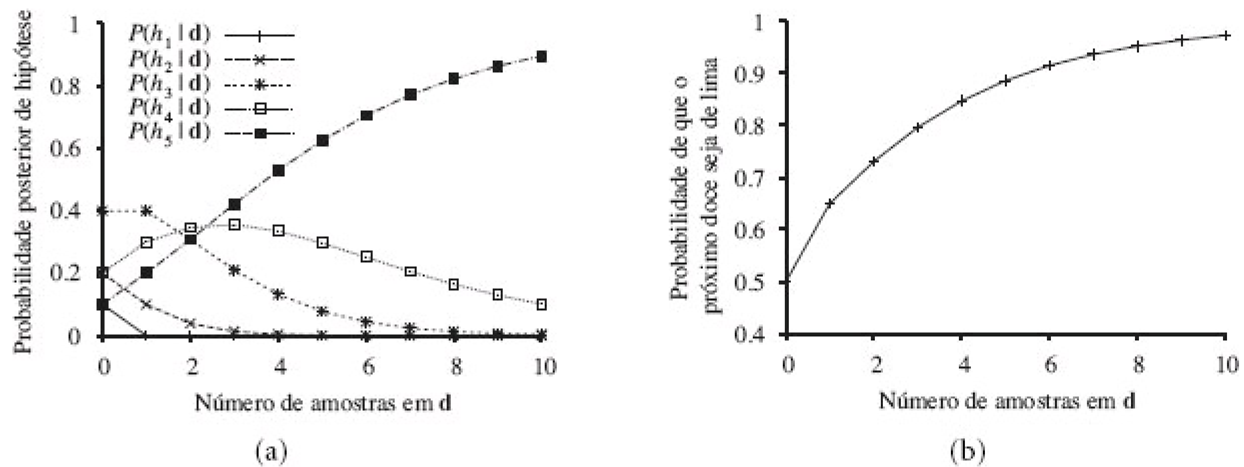


Figura 20.1 (a) Probabilidades *a posteriori* $P(h_i \mid d_1,\ldots, d_n)$ a partir da Equação 20.1. O número de observações N varia de 1-10, e cada observação é de um doce de lima. (b) Previsão bayesiana $P(d_{n+1} = lima \mid d_1,\ldots, d_n)$ a partir da Equação 20.2.

O exemplo mostra que *a previsão bayesiana eventualmente concorda com a verdadeira hipótese*. Isso é característico da aprendizagem bayesiana. Para qualquer probabilidade *a priori* fixa que não elimina a hipótese verdadeira, a probabilidade *a posteriori* de qualquer hipótese falsa vai, a partir de certo ponto, desaparecer sob determinadas condições técnicas. Isso simplesmente acontece porque a probabilidade de gerar dados "não característicos" indefinidamente é muitíssimo pequena (esse ponto é análogo à observação feita na discussão da aprendizagem PAC no Capítulo 18). Mais importante ainda, a previsão bayesiana é *ótima*, quer o conjunto de dados seja pequeno, quer seja grande. Dada a hipótese *a priori*, qualquer outra previsão será correta com menor frequência.

É claro que o caráter ótimo da aprendizagem bayesiana tem um preço. Para problemas reais de aprendizagem, o espaço de hipóteses é em geral muito grande ou infinito, como vimos no Capítulo 18. Em alguns casos, o somatório da Equação 20.2 (ou integração, no caso contínuo) pode ser executado de forma tratável, mas, na maioria dos casos, devemos recorrer a métodos aproximados ou simplificados.

Uma aproximação muito comum — habitualmente adotada na ciência — é fazer previsões com base em uma única hipótese *mais provável*, isto é, uma h_i que maximize $P(h_i \mid \mathbf{d})$. Com frequência, isso é chamado de hipótese de **máximo *a posteriori*** ou MAP. As previsões feitas de acordo com uma hipótese de MAP h_{MAP} são aproximadamente bayesianas até o ponto em que $\mathbf{P}(X|\mathbf{d}) \approx \mathbf{P}(X|h_{\text{MAP}})$. Em nosso exemplo de doces, $h_{\text{MAP}} = h_5$, após três doces de lima seguidos e, assim, o sistema de aprendizagem de MAP prevê que o quarto doce será de lima com probabilidade 1,0 — uma previsão muito mais perigosa que a previsão bayesiana de 0,8 mostrada na Figura 20.1(b). À medida que chegam mais dados, as previsões de MAP e Bayes ficam mais próximas porque os concorrentes da

hipótese de MAP se tornam cada vez menos prováveis.

Embora nosso exemplo não mostre, a descoberta de hipóteses de MAP frequentemente é muito mais fácil que a aprendizagem bayesiana porque exige a resolução de um problema de otimização, em vez de um grande problema de somatório (ou integração). Veremos exemplos desse fato mais adiante no capítulo.

Tanto na aprendizagem bayesiana quanto na aprendizagem de MAP, a hipótese *a priori* $P(h_i)$ desempenha uma função importante. Vimos no Capítulo 18 que a **superadaptação** pode ocorrer quando o espaço de hipóteses é muito expressivo, de forma que ele contenha muitas hipóteses que se ajustam bem ao conjunto de dados. Em vez de impor um limite arbitrário sobre as hipóteses a serem consideradas, os métodos de aprendizagem bayesiana e MAP utilizam a hipótese *a priori* para *penalizar a complexidade*. Em geral, as hipóteses mais complexas têm uma probabilidade *a priori* mais baixa — em parte porque normalmente existem muito mais hipóteses complexas que hipóteses simples. Por outro lado, as hipóteses mais complexas têm capacidade maior de se adaptar aos dados (no caso extremo, uma tabela de busca pode reproduzir os dados exatamente com probabilidade 1). Consequentemente, a hipótese *a priori* incorpora um compromisso entre a complexidade de uma hipótese e seu grau de adaptação aos dados.

Podemos ver o efeito dessa solução de compromisso mais claramente no caso lógico em que H contém apenas hipóteses *determinísticas*. Nesse caso, $P(\mathbf{d} \mid h_i)$ é 1 se h_i é consistente e 0 em caso contrário. Examinando a Equação 20.1, vemos que h_{MAP} será a *teoria lógica mais simples consistente com os dados*. Portanto, uma aprendizagem máxima *a posteriori* fornece uma incorporação natural da navalha de Ockham.

Outra ideia sobre a solução de compromisso entre complexidade e grau de adaptação é obtida tomando-se o logaritmo da Equação 20.1. A escolha de h_{MAP} para maximizar $P(\mathbf{d} \mid h_i)P(h_i)$ é equivalente a minimizar

$$-\log_2 P(\mathbf{d} \mid h_i) - \log_2 P(h_i).$$

Usando a conexão entre codificação de informações e probabilidade que introduzimos na Seção 18.3.4, vemos que o termo $-\log_2 P(h_i)$ é igual ao número de bits necessários para especificar a hipótese h_i. Além disso, $-\log_2 P(\mathbf{d} \mid h_i)$ é o número adicional de bits exigidos para especificar os dados, dada a hipótese (para ver isso, considere que nenhum bit é necessário se a hipótese prevê os dados exatamente — como ocorre com h_5 e a sequência de doces de lima — e que $\log_2 1 = 0$). Consequentemente, a aprendizagem de MAPé a escolha da hipótese que fornece *compactação* máxima dos dados. A mesma tarefa é tratada mais diretamente pelo método de aprendizagem de **comprimento mínimo de descrição**, ou CMD.

Considerando que a aprendizagem MAP expressa simplicidade ao atribuir maiores probabilidades a hipóteses mais simples, CMD expressa isso diretamente pela contagem de bits em uma codificação binária das hipóteses e dos dados.

Uma simplificação final é fornecida supondo-se uma probabilidade *a priori* **uniforme** sobre o espaço de hipóteses. Nesse caso, a aprendizagem de MAP se reduz à escolha de um h_i que maximize

$P(\mathbf{d} \mid h_i)$. Isso é chamado de hipótese de **máxima probabilidade** (MP), h_{MP}. A aprendizagem de máxima probabilidade é muito comum em estatística, uma disciplina na qual muitos pesquisadores desconfiam da natureza subjetiva de hipóteses *a priori*. É uma abordagem razoável quando não existe nenhuma razão para preferir uma hipótese sobre outra *a priori* — por exemplo, quando todas as hipóteses são igualmente complexas. Ela proporciona uma boa aproximação para a aprendizagem bayesiana e de MAP quando o conjunto de dados é grande, porque os dados inundam a distribuição *a priori* sobre hipóteses, mas tem problemas (como veremos) com conjuntos de dados pequenos.

20.2 APRENDIZAGEM COM DADOS COMPLETOS

A **estimativa de densidade** é a tarefa geral de aprender um modelo probabilístico, dados os dados que assumimos ser gerados desse modelo. (A expressão originalmente aplicava-se a funções de densidade de probabilidade para variáveis contínuas, mas agora também é usada para distribuições discretas.)

Esta seção abrange o caso mais simples, no qual temos dados completos. Os dados estão completos quando cada ponto de dado contém valores para cada variável no modelo de probabilidade que está sendo aprendido. Nós nos concentraremos em **aprendizagem de parâmetros** — encontrar os parâmetros numéricos para um modelo de probabilidade cuja estrutura é fixa. Por exemplo, poderíamos estar interessados em aprender as probabilidades condicionais em uma rede bayesiana com determinada estrutura. Vamos também analisar brevemente o problema de estrutura de aprendizagem e de estimativa de densidade não paramétrica.

20.2.1 Aprendizagem de parâmetros de máxima probabilidade: modelos discretos

Vamos supor que compramos um saco de doces de lima e cereja de um novo fabricante cujas proporções de cereja e lima são completamente desconhecidas; a fração poderia estar em qualquer lugar entre 0 e 1. Nesse caso, temos uma quantidade contínua de hipóteses. O **parâmetro**, que chamaremos θ, é a proporção de doces de cereja, e a hipótese é h_θ (a proporção de lima é simplesmente $1 - \theta$). Se supusermos que todas as proporções são igualmente prováveis *a priori*, uma abordagem de máxima probabilidade será razoável. Se modelarmos a situação com uma rede bayesiana, precisaremos de apenas uma variável aleatória, *Sabor* (o sabor de um doce escolhido ao acaso no saco). Ela tem valores *cereja* e *lima*, em que a probabilidade de *cereja* é θ (veja a Figura 20.2(a)). Agora, vamos supor que desembrulhamos N doces, dos quais c são de cereja e $l = N - c$ são de lima. De acordo com a Equação 20.3, a probabilidade desse conjunto de dados específico é:

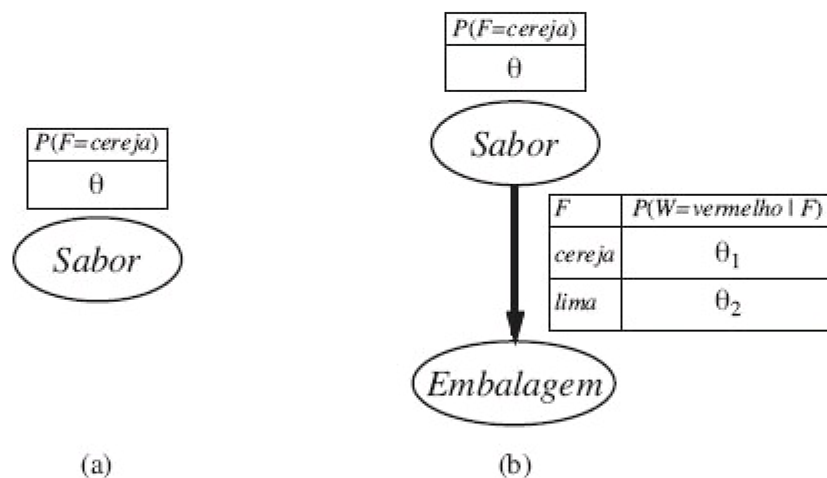


Figura 20.2 (a) Modelo de rede bayesiana para o caso de doces com uma proporção desconhecida de cerejas e limas. (b) Modelo para o caso em que a cor da embalagem depende (probabilisticamente) do sabor do doce.

$$P(\mathbf{d} \mid h_\theta) = \prod_{j=1}^{N} P(d_j \mid h_\theta) = \theta^c \cdot (1-\theta)^\ell \,.$$

A hipótese de máxima probabilidade é dada pelo valor de θ que maximiza essa expressão. O mesmo valor é obtido maximizando-se a **probabilidade logarítmica**,

$$L(\mathbf{d} \mid h_\theta) = \log P(\mathbf{d} \mid h_\theta) = \sum_{j=1}^{N} \log P(d_j \mid h_\theta) = c \log \theta + \ell \log(1-\theta) \,.$$

(Usando logaritmos, reduzimos o produto a uma soma sobre os dados, que em geral é mais fácil de maximizar.) Para encontrar o valor de máxima probabilidade de θ, diferenciamos L em relação a θ e definimos a expressão resultante como zero:

$$\frac{dL(\mathbf{d} \mid h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \qquad \Rightarrow \qquad \theta = \frac{c}{c+\ell} = \frac{c}{N} \,.$$

Então, em linguagem comum, a hipótese de máxima probabilidade h_{MP} afirma que a proporção real de cerejas no saco é igual à proporção observada nos doces desembrulhados até agora!

Parece que tivemos um bocado de trabalho para descobrir o óbvio. Entretanto, na verdade, definimos um método-padrão para aprendizagem de parâmetros, um método com ampla aplicabilidade:

1. Escrever uma expressão para a probabilidade dos dados como uma função do(s) parâmetro(s).
2. Escrever a derivada da probabilidade logarítmica com relação a cada parâmetro.
3. Encontrar os valores de parâmetros tais que as derivadas sejam iguais a zero.

A etapa mais complicada normalmente é a última. Em nosso exemplo, ela foi trivial, mas veremos que, em muitos casos, precisamos recorrer a algoritmos de solução iterativa ou a outras técnicas de otimização numérica, conforme escrevemos no Capítulo 4. O exemplo também ilustra um

problema significativo com a aprendizagem de máxima probabilidade em geral: *quando o conjunto de dados é pequeno o suficiente para que alguns eventos ainda não tenham sido observados — por exemplo, nenhum doce de cereja —, a hipótese de máxima probabilidade atribui a probabilidade zero a esses eventos*. Vários artifícios são usados para evitar esse problema, como inicializar a contagem para cada evento com o valor 1 em vez de zero.

Vamos examinar outro exemplo. Suponha que esse novo fabricante de doces queira dar uma pequena sugestão ao consumidor e utilize embalagens de doces coloridas de vermelho e verde. A *Embalagem* para cada doce é selecionada *probabilisticamente*, de acordo com alguma distribuição condicional desconhecida, dependendo do sabor. O modelo de probabilidade correspondente é mostrado na Figura 20.2(b). Note que ele tem três parâmetros: θ, θ_1 e θ_2. Com esses parâmetros, a probabilidade de ver, digamos, um doce de cereja em uma embalagem verde pode ser obtida a partir da semântica-padrão para redes bayesianas:

$$\begin{aligned}&P(Sabor = cereja, Embalagem = verde \mid h_{\theta,\theta_1,\theta_2})\\&= P(Sabor = cereja \mid h_{\theta,\theta_1,\theta_2})\, P(Embalagem = verde \mid Sabor = cereja, h_{\theta,\theta_1,\theta_2})\\&= \theta \cdot (1 - \theta_1).\end{aligned}$$

Agora, desembrulhamos N doces, dos quais c são cerejas e l são limas. As contagens de embalagens são: r_c dos doces de cereja têm embalagens vermelhas e g_c têm embalagens verdes, enquanto r_l dos doces de lima têm embalagens vermelhas e g_l têm embalagens verdes. A probabilidade dos dados é fornecida por:

$$P(\mathbf{d} \mid h_{\theta,\theta_1,\theta_2}) = \theta^c(1-\theta)^\ell \cdot \theta_1^{r_c}(1-\theta_1)^{g_c} \cdot \theta_2^{r_\ell}(1-\theta_2)^{g_\ell} \,.$$

Essa expressão parece horrível, mas o uso de logaritmos ajuda:

$$L = [c\log\theta + \ell\log(1-\theta)] + [r_c\log\theta_1 + g_c\log(1-\theta_1)] + [r_\ell\log\theta_2 + g_\ell\log(1-\theta_2)] \,.$$

A vantagem de usar logs é clara: a probabilidade logarítmica é a soma de três termos, cada um dos quais contém um único parâmetro. Quando tomarmos derivadas em relação a cada parâmetro e as definirmos como zero, conseguiremos três equações independentes, cada uma contendo apenas um parâmetro:

$$\begin{aligned}\frac{\partial L}{\partial\theta} &= \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 &&\Rightarrow \quad \theta = \frac{c}{c+\ell}\\ \frac{\partial L}{\partial\theta_1} &= \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 &&\Rightarrow \quad \theta_1 = \frac{r_c}{r_c+g_c}\\ \frac{\partial L}{\partial\theta_2} &= \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0 &&\Rightarrow \quad \theta_2 = \frac{r_\ell}{r_\ell+g_\ell}\,.\end{aligned}$$

A solução para θ é a mesma de antes. A solução para θ_1, a probabilidade de que um doce de cereja tenha embalagem vermelha, é a fração observada de doces de cereja com embalagens vermelhas, e o mesmo ocorre no caso de θ_2.

☞ Esses resultados são muito confortantes, e é fácil ver que eles podem ser estendidos a qualquer rede bayesiana cujas probabilidades condicionais são representadas como tabelas. O ponto mais importante é que, *com dados completos, o problema de aprendizagem de parâmetros de máxima probabilidade para uma rede bayesiana se decompõe em problemas de aprendizagem separados,*

um para cada parâmetro (veja o Exercício 20.6 para o caso não tabulado, em que cada parâmetro afeta várias probabilidades condicionais). O segundo ponto importante é que os valores de parâmetros para uma variável, dados seus pais, são apenas as frequências observadas dos valores de variáveis para cada configuração dos valores dos pais. Como antes, devemos ser cuidadosos para evitar zeros quando o conjunto de dados é pequeno.

20.2.2 Modelos de Bayes ingênuos

Provavelmente, o modelo de rede bayesiana mais comum utilizado na aprendizagem de máquina é o modelo de **Bayes ingênuo**. Nesse modelo, a variável de "classe" C (que deve ser prevista) é a raiz, e as variáveis de "atributos" Xi são as folhas. O modelo é "ingênuo" porque supõe que os atributos são condicionalmente independentes uns dos outros, dada a classe (o modelo da Figura 20.2(b) é um modelo de Bayes ingênuo com a classe *Sabor* e apenas um atributo, *Embalagem*). Supondo-se variáveis booleanas, os parâmetros são:

$$\theta = P(C = verdadeiro),\ \theta_{i1} = P(X_i = verdadeiro \mid C = verdadeiro),\ \theta_{i2} = P(X_i = verdadeiro \mid C = falso).$$

Os valores de parâmetros de máxima probabilidade são encontrados exatamente do mesmo modo que aparece na Figura 20.2(b). Uma vez que o modelo é treinado dessa maneira, ele pode ser utilizado para classificar novos exemplos, para os quais a variável de classe C é não observada. Com valores de atributos observados $x_1, \ldots, x_n$, a probabilidade de cada classe é dada por:

$$\mathbf{P}(C \mid x_1, \ldots, x_n) = \alpha\, \mathbf{P}(C) \prod_i \mathbf{P}(x_i \mid C)\,.$$

Uma previsão determinística pode ser obtida escolhendo-se a classe mais provável. A Figura 20.3 mostra a curva de aprendizagem para esse método quando ele é aplicado ao problema de restaurante do Capítulo 18. O método aprende bastante bem, mas não tanto quanto a aprendizagem de árvore de decisão; presumivelmente, isso ocorre porque a hipótese verdadeira — que é uma árvore de decisão — não pode ser representada exatamente com o uso de um modelo de Bayes ingênuo. A aprendizagem bayesiana ingênua acaba funcionando surpreendentemente bem em ampla variedade de aplicações; a versão reforçada (Exercício 20.4) é um dos mais efetivos algoritmos de aprendizagem de uso geral. A aprendizagem bayesiana ingênua se adapta bem à escala de problemas muito grandes: com n atributos booleanos, existem apenas $2n + 1$ parâmetros e *nenhuma busca é necessária para encontrar* h_{ML}*, a hipótese de Bayes ingênua de máxima probabilidade*. Por fim, a aprendizagem bayesiana ingênua não tem nenhuma dificuldade com dados ruidosos ou faltantes e pode fornecer previsões probabilísticas quando apropriado.

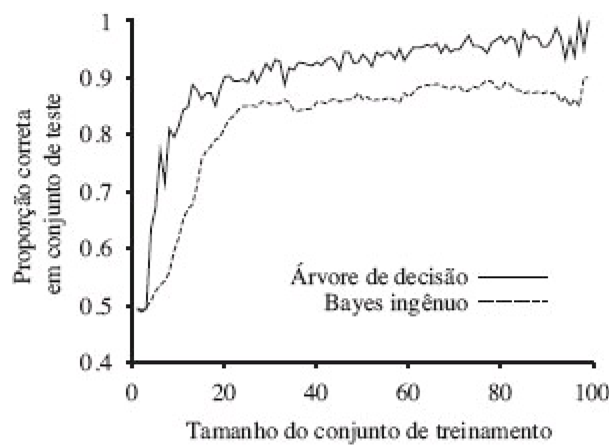


Figura 20.3 A curva de aprendizagem correspondente à aprendizagem bayesiana ingênua aplicada ao problema de restaurante do Capítulo 18; a curva de aprendizagem para a aprendizagem em árvore de decisão é mostrada para comparação.

20.2.3 Aprendizagem de parâmetros de máxima probabilidade: modelos contínuos

Os modelos de probabilidade contínuos, como o modelo **gaussiano linear**, foram introduzidos na Seção 14.3. Pelo fato de as variáveis contínuas serem onipresentes em aplicações do mundo real, é importante saber como aprender modelos contínuos a partir de dados. Os princípios para a aprendizagem de máxima probabilidade são idênticos aos casos discreto e contínuos.

Vamos começar com um caso muito simples: a aprendizagem dos parâmetros de uma função de densidade gaussiana sobre uma única variável. Isto é, os dados são gerados como a seguir:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} .$$

Os parâmetros desse modelo são a média *m* e o desvio-padrão *s* (note que a "constante" de normalização depende de *s* e, assim, não podemos ignorá-la). Sejam os valores observados $x_1, \ldots, x_n$. Então, a probabilidade logarítmica é:

$$L = \sum_{j=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} = N(-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1}^{N} \frac{(x_j-\mu)^2}{2\sigma^2} .$$

Como sempre, definindo as derivadas como zero, obtemos:

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= -\frac{1}{\sigma^2}\sum_{j=1}^{N}(x_j-\mu) = 0 &&\Rightarrow \quad \mu = \frac{\sum_j x_j}{N} \\ \frac{\partial L}{\partial \sigma} &= -\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{N}(x_j-\mu)^2 = 0 &&\Rightarrow \quad \sigma = \sqrt{\frac{\sum_j (x_j-\mu)^2}{N}} . \end{aligned} \qquad (20.4)$$

Isto é, o valor de máxima probabilidade da média é a média das amostras, e o valor de máxima probabilidade do desvio-padrão é a raiz quadrada da variância das amostras. Novamente, esses são resultados reconfortantes que confirmam a prática de "senso comum".

Agora, considere um modelo gaussiano linear com um pai contínuo X e um filho contínuo Y. Conforme já explicamos, Y tem uma distribuição gaussiana cuja média depende linearmente do valor d e X e cujo desvio-padrão é fixo. Para aprender a distribuição condicional $P(Y \mid X)$, podemos maximizar a probabilidade condicional:

$$P(y \mid x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-(\theta_1 x+\theta_2))^2}{2\sigma^2}} . \tag{20.5}$$

Aqui, os parâmetros são θ_1, θ_2 e s. Os dados são uma coleção de pares (x_j, y_j), como ilustra a Figura 20.4. Usando os métodos habituais (Exercício 20.5), podemos encontrar os valores de máxima probabilidade dos parâmetros. Aqui o ponto é diferente. Se considerarmos apenas os parâmetros θ_1 e θ_2 que definem o relacionamento linear entre x e y, fica claro que maximizar a probabilidade logarítmica com relação a esses parâmetros é o mesmo que *minimizar* o numerador $(y - (\theta_1 x + \theta_2))^2$ no expoente da Equação 20.5. Isso é a perda L_2, o erro quadrático entre o valor real y e a previsão $\theta_1 x + \theta_2$. Essa é a quantidade minimizada pelo procedimento-padrão de **regressão linear** descrita na Seção 18.6. Agora, podemos compreender por que: a minimização da soma de erros quadráticos fornece o modelo de linha reta de máxima probabilidade, desde que os dados sejam gerados com ruído gaussiano de variância fixa.

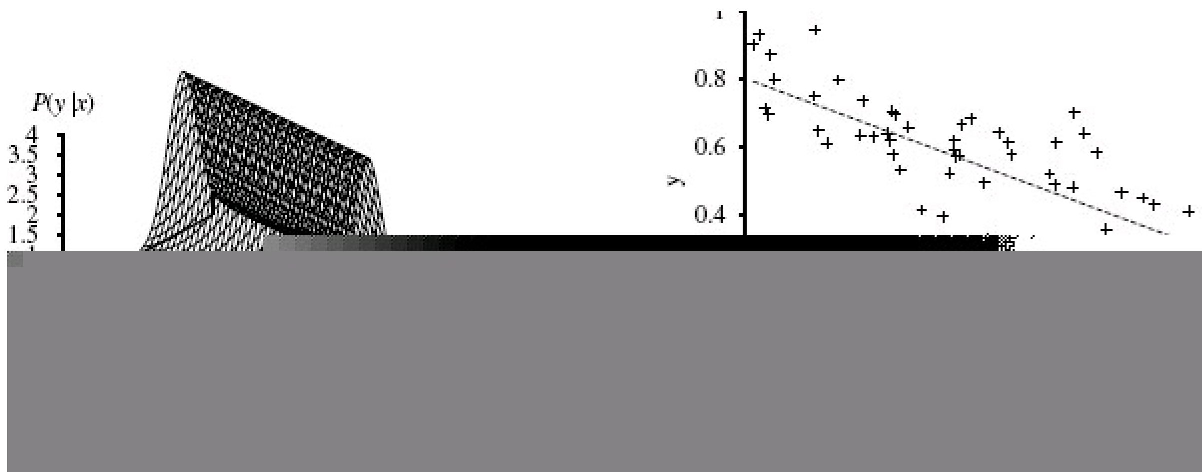


Figura 20.4 (a) Modelo gaussiano linear descrito como $y = \theta_1 x + \theta_2$ somado ao ruído gaussiano com variância fixa. (b) Conjunto de 50 pontos de dados gerados a partir desse modelo.

20.2.4 Aprendizagem de parâmetros bayesiana

A aprendizagem de máxima probabilidade dá origem a alguns procedimentos muito simples, mas tem algumas deficiências sérias com conjuntos de dados pequenos. Por exemplo, depois de ver um doce de cereja, a hipótese de máxima probabilidade é que o saco seja 100% de doces de cereja (ou seja, $\theta = 1{,}0$). A menos que a hipótese *a priori* de alguém seja a de que os sacos devem ser totalmente de cereja ou de lima, essa não é uma conclusão razoável. O mais provável é que o saco seja uma mistura de cereja e lima. A abordagem bayesiana para aprendizagem de parâmetros começa pela definição de uma distribuição de probabilidade prévia sobre a hipóteses possível. Chamamos

isso de **hipótese *a priori***. Então, à medida que os dados chegam, a distribuição de probabilidade posterior é atualizada.

O exemplo de doce da Figura 20.2(a) tem um único parâmetro, θ: a probabilidade de um pedaço de doce selecionado ao acaso ter sabor de cereja. Na visão de Bayes, θ é o valor (desconhecido) de uma variável aleatória Θ que define o espaço de hipótese; a hipótese *a priori* é simplesmente a distribuição *a priori* $\mathbf{P}(\Theta)$. Desse modo, $P(\Theta = \theta)$ é a probabilidade *a priori* de que o saco contenha uma fração θ de doces de cereja.

Se o parâmetro θ puder ter qualquer valor entre 0 e 1, então $\mathbf{P}(\Theta)$ deve ser uma distribuição contínua diferente de zero apenas entre 0 e 1 e que tem integral 1. A densidade uniforme $P(\theta) = \mathit{Uniforme}[0,1](\theta)$ é uma candidata (veja o Capítulo 13). Ocorre que a densidade uniforme é um membro da família de **distribuições beta**. Cada distribuição beta é definida por dois **hiperparâmetros**[3] a e b, tais que

$$\text{beta}[a,b](\theta) = \alpha\, \theta^{a-1}(1-\theta)^{b-1}\,, \tag{20.6}$$

para θ no intervalo [0, 1]. A constante de normalização a, que faz com que a distribuição se integre com 1, depende de a e b (veja o Exercício 20.7). A Figura 20.5 mostra qual é a aparência da distribuição para diversos valores de a e b. O valor médio da distribuição é $a/(a + b)$ e, assim, valores maiores de a sugerem a crença de que Θ está mais próxima de 1 do que de 0. Valores maiores de $a + b$ dão à distribuição um pico mais estreito, sugerindo maior certeza sobre o valor de Θ. Desse modo, a família beta fornece uma faixa útil de possibilidades para a hipótese *a priori*.

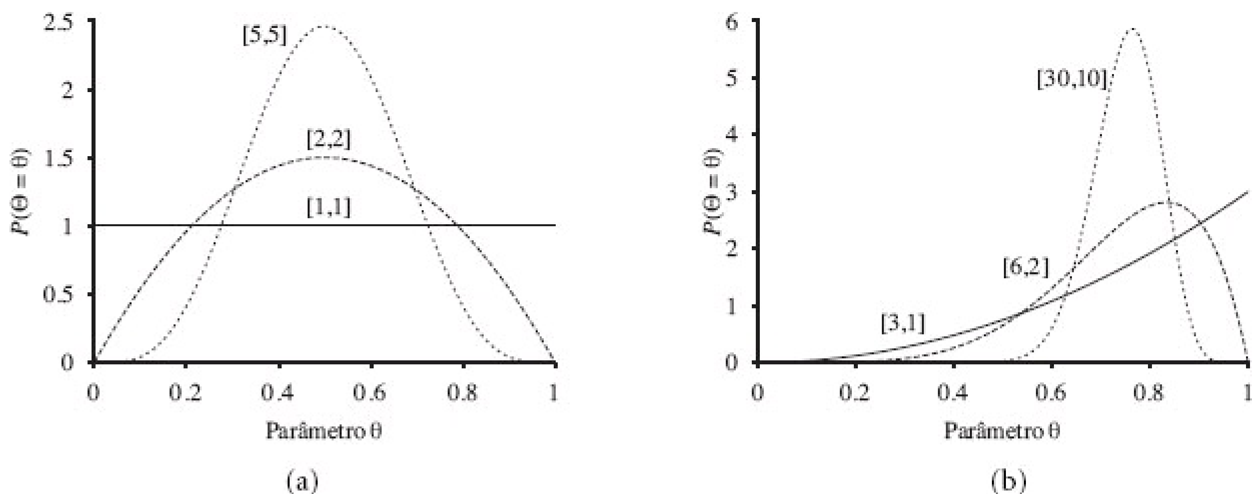


Figura 20.5 Exemplos da distribuição beta[a, b] para diferentes valores de [a, b].

Além de sua flexibilidade, a família beta tem outra propriedade interessante: se Θ tem uma probabilidade *a priori* beta[a, b], então, depois da observação de um ponto de dados, a distribuição posterior para Θ também é uma distribuição beta. Em outras palavras, beta é fechado sob atualização. A família beta é chamada **conjugado *a priori*** para a família de distribuições correspondente a uma variável booleana.[4] Vejamos como isso funciona. Vamos supor que observamos um doce de cereja; então, temos

$$\begin{aligned} P(\theta \mid D_1 = cereja\ ) &= \alpha\, P(D_1 = cereja\ \mid \theta)P(\theta) \\ &= \alpha'\ \theta \cdot \text{beta}[a,b](\theta) = \alpha'\ \theta \cdot \theta^{a-1}(1-\theta)^{b-1} \\ &= \alpha'\ \theta^{a}(1-\theta)^{b-1} = \text{beta}[a+1,b](\theta)\ . \end{aligned}$$

Desse modo, depois de ver um doce de cereja, simplesmente incrementamos o parâmetro a para obter a posterior; de modo semelhante, depois de ver um doce de lima, incrementamos o parâmetro b. Portanto, podemos visualizar os hiperparâmetros a e b como **contagens virtuais**, no sentido de que um *a priori* beta[a, b] se comporta exatamente como se tivéssemos começado com um *a priori* uniforme beta[1, 1] e visto a – 1 doces de cereja reais e b – 1 doces de lima reais.

Examinando uma sequência de distribuições beta para valores crescentes de a e b, mantendo as proporções fixas, podemos ver nitidamente como a distribuição posterior sobre o parâmetro Θ se altera à medida que os dados chegam. Por exemplo, suponha que o saco de doces real tenha 75% de cereja. A Figura 20.5(b) mostra a sequência beta[3, 1], beta[6, 2], beta[30, 10]. É claro que a distribuição está convergindo para um pico estreito em torno do valor verdadeiro de Θ. Então, para grandes conjuntos de dados, a aprendizagem bayesiana (pelo menos nesse caso) converge para dar os mesmos resultados que a aprendizagem de máxima probabilidade.

Consideremos agora um caso mais complicado. A rede da Figura 20.2(b) tem três parâmetros, θ, θ_1 e θ_2, onde θ_1 é a probabilidade de embalagem vermelha em um doce de cereja e θ_2 é a probabilidade de embalagem vermelha em um doce de lima. A hipótese de Bayes anterior deve cobrir todos os três parâmetros, isto é, precisamos especificar $\mathbf{P}(\Theta, \Theta_1, \Theta_2)$. Usualmente, supomos **independência de parâmetros**:

$$\mathbf{P}(\Theta, \Theta_1, \Theta_2) = \mathbf{P}(\Theta)\mathbf{P}(\Theta_1)\mathbf{P}(\Theta_2)\ .$$

Com essa suposição, cada parâmetro pode ter sua própria distribuição beta que é atualizada separadamente à medida que os dados chegam. A Figura 20.6 mostra como podemos incorporar a hipótese *a priori* e qualquer dado em uma rede bayesiana. Os nós Θ, Θ_1, Θ_2, não têm pais. Mas, a cada vez que fazemos uma observação de embalagem e sabor correspondente a um pedaço de doce, adicionamos um nó $Sabor_i$, que é dependente do parâmetro sabor Θ:

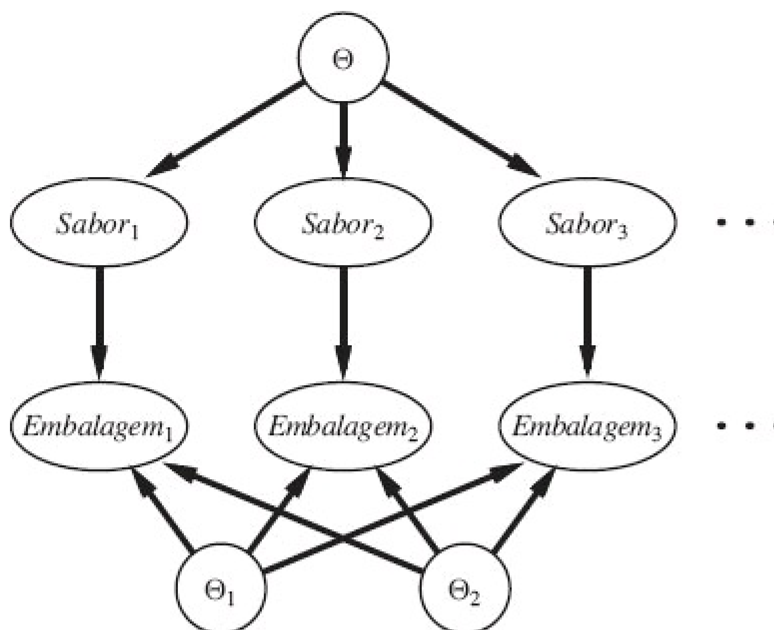


Figura 20.6 Uma rede bayesiana que corresponde a um processo de aprendizagem bayesiana. Distribuições posteriores para as variáveis de parâmetros Θ, Θ_1 e Θ_2 podem ser deduzidas de suas distribuições *a priori* e da evidência nas variáveis $Sabor_i$ e $Embalagem_i$.

$$P(Sabor_i = cereja \mid \Theta = \theta) = \theta.$$

Adicionamos também um nó $Embalagem_i$, que é dependente de Θ_1 e Θ_2:

$$P(Embalagem_i = \text{vermelho} \mid Sabor_i = cereja, \Theta_1 = \theta_1) = \theta_1$$
$$P(Embalagem_i = \text{vermelho} \mid Sabor_i = lima, \Theta_2 = \theta_2) = \theta_2.$$

Agora, todo o processo de aprendizagem bayesiana pode ser formulado como um problema de *inferência.* Adicionamos nós de evidência novos, em seguida consultamos os nós desconhecidos (nesse caso, Θ, Θ_1, Θ_2). Essa formulação de aprendizagem e previsão deixa claro que a aprendizagem bayesiana não requer "princípios de aprendizagem" extras. Além disso, *em essência, existe apenas um algoritmo de aprendizagem*, ou seja, o algoritmo de inferência para redes bayesianas. É claro, a natureza dessas redes é um pouco diferente daquelas do Capítulo 14, por causa do número potencialmente grande de variáveis de evidência que representam o conjunto de treinamento e por causa da prevalência de parâmetros variáveis de valor contínuo.

20.2.5 Aprendizagem de estruturas de redes bayesianas

Até agora, supomos que a estrutura da rede bayesiana é dada, e estamos apenas tentando aprender os parâmetros. A estrutura da rede representa o conhecimento causal básico sobre o domínio que frequentemente é fácil de fornecer para um usuário especialista ou mesmo para um usuário ingênuo. Porém, em alguns casos, o modelo causal pode estar indisponível ou sujeito a disputa — por exemplo, certas corporações afirmaram há muito tempo que fumar não causa câncer —, assim é importante entender como a estrutura de uma rede bayesiana pode ser aprendida a partir dos dados. Esta seção apresenta um esquema breve das ideias principais.

A abordagem mais óbvia é *buscar* um bom modelo. Podemos iniciar com um modelo que não contém nenhum vínculo e começar a adicionar pais correspondentes a cada nó, ajustando os parâmetros com os métodos que acabamos de examinar e medindo a exatidão do modelo resultante. Como alternativa, podemos começar com um palpite inicial sobre a estrutura e utilizar a busca de subida de encosta ou a busca de têmpera simulada para fazer modificações, retornando os parâmetros após cada mudança na estrutura. As modificações podem incluir inversão, adição ou eliminação de arcos. Não devemos introduzir ciclos no processo, pois muitos algoritmos pressupõem que é dada uma ordenação para as variáveis e que um nó pode ter pais somente entre os nós que vêm antes deles na ordenação (exatamente como no processo de construção do Capítulo 14). Para generalizar, também precisamos fazer uma busca sobre ordenações possíveis.

Existem dois métodos alternativos para decidir quando uma boa estrutura foi encontrada. O primeiro é testar se as asserções de independência condicional implícitas na estrutura são realmente satisfeitas nos dados. Por exemplo, o uso de um modelo bayesiano ingênuo para o problema de

restaurante supõe que

$$\mathbf{P}(Sex/Sáb, Bar \mid VaiEsperar) = \mathbf{P}(Sex/Sáb \mid VaiEsperar) = \mathbf{P}(Bar \mid VaiEsperar)$$

e podemos verificar nos dados que a mesma equação é válida entre as frequências condicionais correspondentes. Agora, ainda que a estrutura descreva a verdadeira natureza causal do domínio, flutuações estatísticas no conjunto de dados significam que a equação nunca será satisfeita *exatamente*, e então precisamos executar um teste estatístico apropriado para verificar se existe evidência suficiente de que a hipótese de independência foi violada. A complexidade da rede resultante dependerá do limite usado para esse teste — quanto mais rígido for o teste de independência, mais vínculos serão adicionados e maior será o perigo de superadaptação.

Uma abordagem mais consistente com as ideias deste capítulo é avaliar até que ponto o modelo proposto explica os dados (em sentido probabilístico). No entanto, devemos ser cuidadosos com a forma como efetuamos essa medição. Se tentarmos simplesmente encontrar a hipótese de máxima probabilidade, acabaremos com uma rede completamente conectada porque a adição de outros pais a um nó não poderá diminuir a probabilidade (Exercício 20.8). Somos forçados a penalizar a complexidade do modelo de algum modo. A abordagem de MAP (ou de CMD) simplesmente subtrai uma penalidade da probabilidade de cada estrutura (depois do ajuste de parâmetros) antes de comparar estruturas diferentes. A abordagem de Bayes coloca uma probabilidade *a priori* conjunta sobre estruturas e parâmetros. Em geral, existem muitas estruturas para se efetuar o somatório (uma quantidade superexponencial em relação ao número de variáveis) e, assim, a maioria dos especialistas utiliza o CMMC para obter amostras sobre estruturas.

Penalizar a complexidade (por métodos de MAP ou de Bayes) introduz uma conexão importante entre a estrutura ótima e a natureza da representação para as distribuições condicionais na rede. Com distribuições tabulares, a penalidade de complexidade correspondente à distribuição de um nó cresce exponencialmente com o número de pais; porém, digamos, com distribuições OU-ruidosas, ela cresce de forma apenas linear. Isso significa que a aprendizagem com modelos de OU-ruidoso (ou com outros modelos parametrizados de modo compacto) tende a produzir estruturas aprendidas com mais pais do que a aprendizagem com distribuições tabulares.

20.2.6 Estimativa de densidade com modelos não paramétricos

É possível aprender um modelo de probabilidade sem fazer suposições sobre sua estrutura e parametrização pela adoção dos métodos não paramétricos da Seção 18.8. A tarefa de **estimativa de densidade não paramétrica** é realizada normalmente em domínios contínuos, como mostrado na Figura 20.7(a). A figura mostra uma função de densidade de probabilidade em um espaço definido por duas variáveis contínuas. Na Figura 20.7(b), vemos uma amostra de pontos de dados a partir dessa função de densidade. A questão é se podemos recuperar o modelo das amostras.

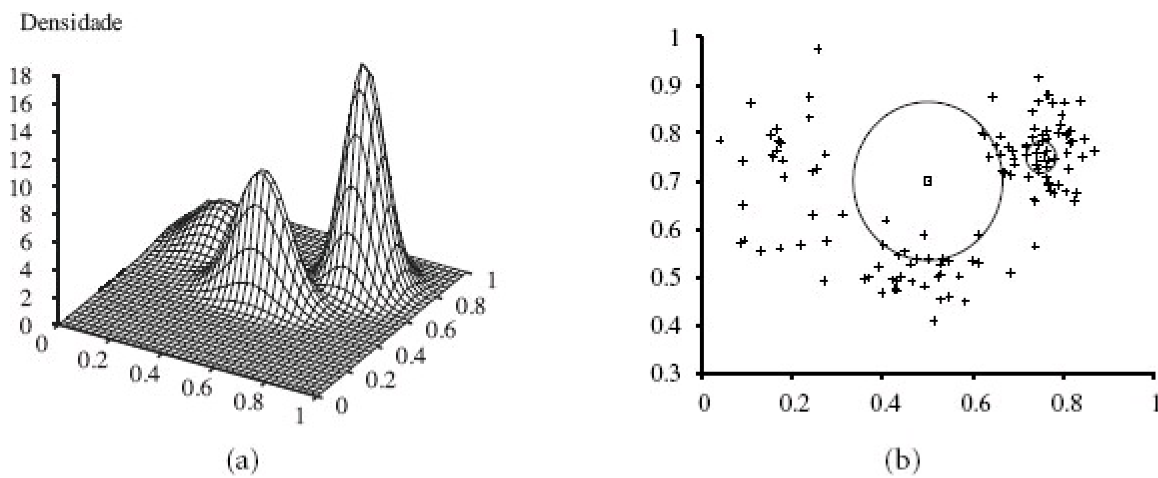


Figura 20.7 (a) Plano em 3-D da mistura das gaussianas da Figura 20.11(a). (b) Amostra de pontos de 128 pontos da mistura, junto com dois pontos de consulta (quadrados pequenos) e seus 10 vizinhos mais próximos (círculos médios e grandes).

Primeiro vamos considerar os modelos de *k*-**vizinhos mais próximos** (no Capítulo 18 vimos modelos de vizinhos mais próximos para classificação e regressão; aqui os vemos para a estimativa de densidade).

Dada uma amostra de pontos de dados, para estimar a densidade de probabilidade desconhecida em um ponto **x** de consulta, podemos simplesmente medir a densidade dos pontos de dados na vizinhança de **x**. A Figura 20.7(b) mostra dois pontos de consulta (pequenos quadrados). Para cada ponto de consulta desenhamos o menor círculo que envolve 10 vizinhos — os 10 vizinhos mais próximos. Podemos ver que o círculo central é maior, significando que lá existe baixa densidade, e o círculo da direita é menor, significando que lá existe alta densidade. Na Figura 20.8 mostramos três planos de estimativa de densidade utilizando *k* vizinhos mais próximos, para diferentes valores de *k*. Parece claro que (b) tende para a direita, enquanto (a) é muito "pontudo" (*k* é muito pequeno) e (c) é muito "liso" (*k* é muito grande).

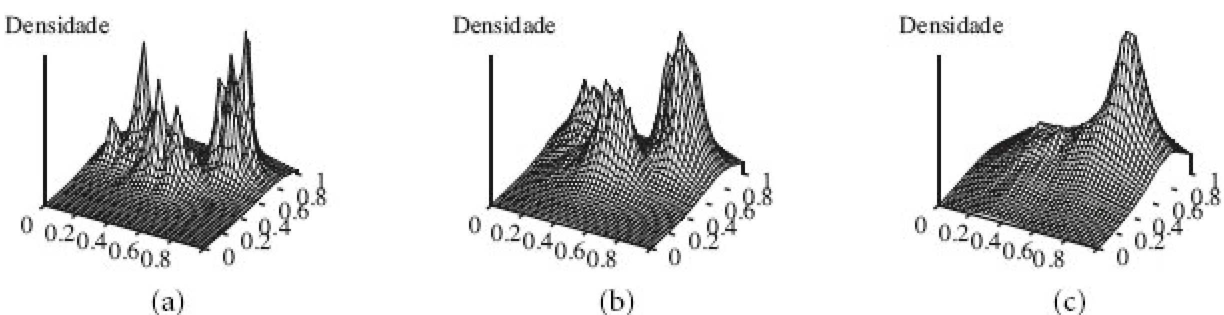


Figura 20.8 Estimativa de densidade utilizando *k* vizinhos mais próximos, aplicados aos dados na Figura 20.7(b), para *k* =3, 10 e 40, respectivamente. *k* = 3 é muito espinhoso, 40 é muito lis, e 10 tende apenas para a direita. O melhor valor de *k* pode ser escolhido por validação cruzada.

Outra possibilidade é usar as **funções de kernel**, como fizemos para a regressão localmente ponderada. Para aplicar um modelo de kernel para estimar a densidade, vamos supor que cada ponto de dados gere a sua própria função de densidade pequena usando um kernel gaussiano. A densidade estimada em um ponto de consulta **x** é, então, a densidade média como dado por cada função do kernel:

$$P(\mathbf{x}) = \frac{1}{N}\sum_{j=1}^{N}\mathcal{K}(\mathbf{x}, \mathbf{x}_j) .$$

Vamos supor gaussianas esféricas com desvio-padrão *w* ao longo de cada eixo:

$$\mathcal{K}(\mathbf{x}, \mathbf{x}_j) = \frac{1}{(w^2\sqrt{2\pi})^d} e^{-\frac{D(\mathbf{x},\mathbf{x}_j)^2}{2w^2}} ,$$

onde *d* é o número de dimensões em **x** e *D* é a função de distância euclidiana. Temos ainda o problema de escolher um valor adequado para a largura *w* do kernel; a Figura 20.9 mostra valores que são muito pequenos, apenas para a direita, e muito extensos. Um bom valor de *w* pode ser escolhido por meio de validação cruzada.

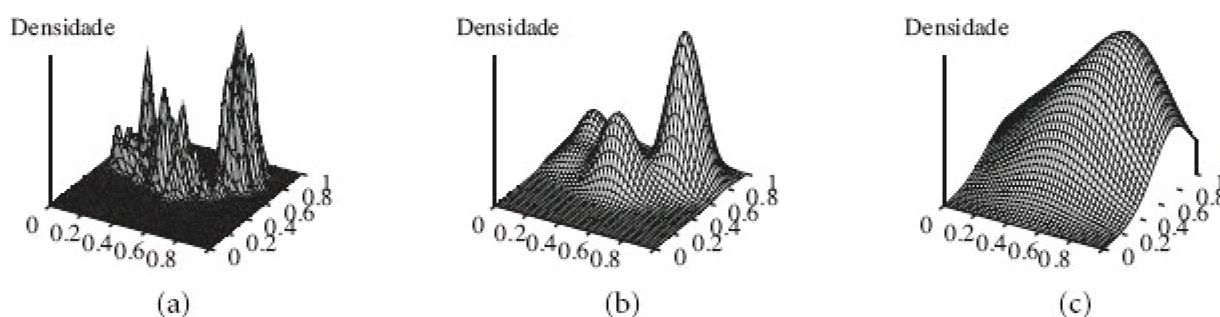


Figure 20.9 Estimativa de densidade de kernel para os dados na Figura 20.7(b), utilizando kernels gaussianos com *w* = 0,02, 0,07 e 0,20, respectivamente *w* = 0,07 tende para a direita.

20.3 APRENDIZAGEM COM VARIÁVEIS OCULTAS: O ALGORITMO EM

A seção precedente lidou com o caso completamente observável. Muitos problemas reais têm **variáveis ocultas** (às vezes chamadas **variáveis latentes**) que não são observáveis nos dados disponíveis para aprendizagem. Por exemplo, registros médicos com frequência incluem os sintomas observados, o tratamento aplicado e, talvez, o resultado do tratamento, mas raramente contêm uma observação direta da própria doença! (Observe que o *diagnóstico* não é a *doença;* é uma consequência causal dos sintomas observados que, por sua vez, são causados pela doença.) Você poderia perguntar: “Se a doença não é observada, por que não construir um modelo sem ela?” A resposta aparece na Figura 20.10, que mostra um pequeno modelo de diagnóstico fictício para doenças do coração. Existem três fatores de predisposição observáveis e três sintomas observáveis (que são deprimentes demais para se identificar). Suponha que cada variável tenha três valores possíveis (por exemplo, nenhum, moderado e severo). A remoção da variável oculta a partir da rede em (a) produz a rede em (b); o número total de parâmetros aumenta de 78 para 708. Desse modo, *variáveis latentes podem reduzir drasticamente o número de parâmetros exigidos para especificar uma rede bayesiana*. Por sua vez, isso pode reduzir drasticamente a quantidade de dados necessários para se aprender os parâmetros.

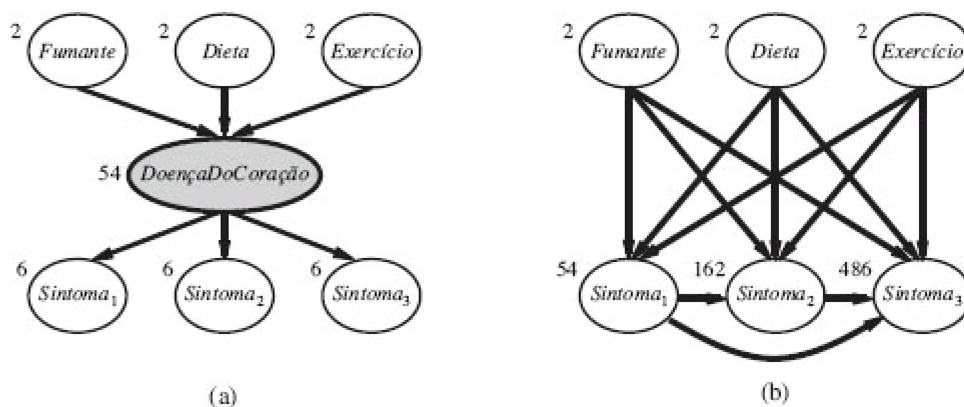


Figura 20.10 (a) Rede de diagnóstico simples para doença do coração, que se supõe ser uma variável oculta. Cada variável tem três valores possíveis e é identificada com o número de parâmetros independentes em sua distribuição condicional; o número total é 78. (b) Rede equivalente c o m *DoençaDoCoração* removida. Observe que as variáveis de sintomas não são mais condicionalmente independentes, dados seus pais. Essa rede exige 708 parâmetros.

As variáveis ocultas são importantes, mas complicam o problema de aprendizagem. Por exemplo, n a Figura 20.10(a), não é óbvia a maneira de aprender a distribuição condicional para *DoençaDoCoração*, dados seus pais, porque não conhecemos o valor de *DoençaDoCoração* em cada caso; o mesmo problema surge na aprendizagem das distribuições correspondentes aos sintomas. Esta seção descreve um algoritmo chamado **maximização de expectativa** (EM), que resolve esse problema de modo muito geral. Mostraremos três exemplos e depois forneceremos uma descrição geral. A princípio, o algoritmo parece mágica; porém, uma vez desenvolvida a intuição, podemos encontrar aplicações para EM em enorme variedade de problemas de aprendizagem.

20.3.1 Formação não supervisionada de agrupamentos: aprendizagem de misturas de gaussianos

A **formação não supervisionada de agrupamentos** é o problema de distinguir várias categorias em uma coleção de objetos. O problema é não supervisionado porque os rótulos de categorias não são dados. Por exemplo, vamos supor que registramos o espectro de centenas de milhares de estrelas; existem diferentes *tipos* de estrelas revelados pelo espectro? Nesse caso, quantos tipos e quais são suas características? Todos nós estamos familiarizados com expressões como "gigante vermelha" e "anã branca", mas as estrelas não têm esses rótulos para identificá-las — os astrônomos tiveram de executar a formação de agrupamentos não supervisionados para identificar essas categorias. Outros exemplos incluem a identificação de espécies, gêneros, ordens, e assim por diante, na taxonomia de organismos de Lineu e a criação de espécies naturais para categorizar objetos comuns (veja o Capítulo 12).

A formação de agrupamentos não supervisionados começa com dados. A Figura 20.11(b) mostra 500 pontos de dados, cada um dos quais especifica os valores de dois atributos contínuos. Os pontos de dados poderiam corresponder a estrelas, e os atributos poderiam corresponder a intensidades espectrais em duas frequências específicas. Em seguida, precisamos compreender que espécie de distribuição de probabilidade poderia ter gerado os dados. A formação de agrupamentos pressupõe

que os dados são gerados a partir de uma **distribuição de mistura** P. Tal distribuição tem k **componentes**, cada um dos quais é por si só uma distribuição. Um ponto de dados é gerado escolhendo-se primeiro um componente e depois gerando-se uma amostra a partir desse componente. Seja a variável aleatória C que denota o componente, com valores 1,…, k; então, a distribuição de mistura é dada por:

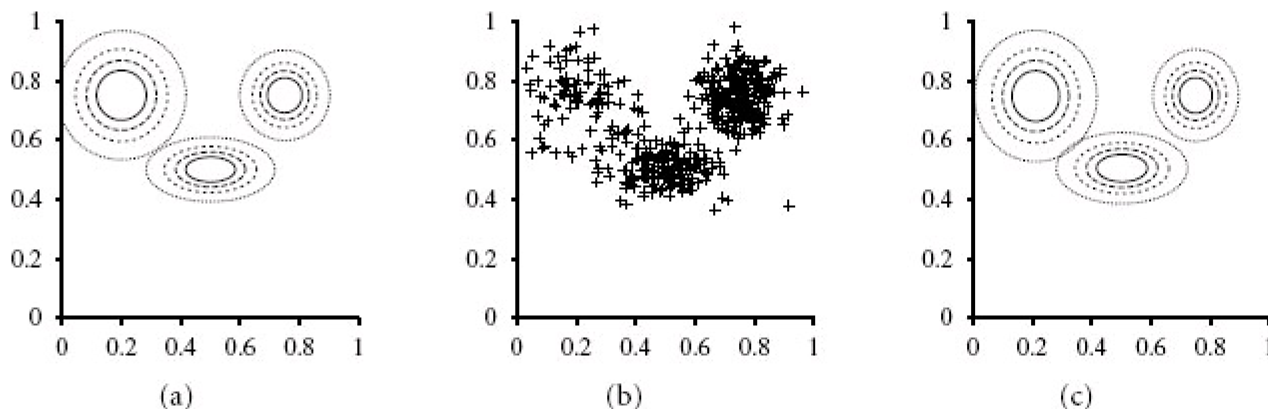


Figura 20.11 (a) Modelo de mistura gaussiana com três componentes; os pesos (da esquerda para a direita) são 0,2, 0,3 e 0,5. (b) 500 pontos de dados da amostra do modelo em (a). (c) Modelo reconstruído por EM a partir dos dados em (b).

$$P(\mathbf{x}) = \sum_{i=1}^{k} P(C=i)\ P(\mathbf{x} \mid C=i)\ ,$$

onde **x** se refere aos valores dos atributos para um ponto de dados. No caso de dados contínuos, uma escolha natural para as distribuições de componentes é a gaussiana multivariada, que fornece a família de distribuições chamada **mistura de distribuições gaussianas**. Os parâmetros de uma mistura de distribuições gaussianas são $w_i = P(C = i)$ (o peso de cada componente), μ_i (a média de cada componente) e Σ_i (a covariância de cada componente). A Figura 20.11(a) mostra uma mistura de três gaussianos; essa mistura é de fato a origem dos dados contidos em (b), assim como o modelo mostrado na Figura 20.7(a).

Então, o problema de formação não supervisionada de agrupamentos consiste em recuperar um modelo de mistura como o da Figura 20.11(b) a partir de dados brutos como os da Figura 20.11(a). É claro que, se *soubéssemos* que componente gerou cada ponto de dados, seria fácil recuperar os gaussianos componentes: poderíamos simplesmente selecionar todos os pontos de dados a partir de um dado componente e depois aplicar (em uma versão multivariada) a Equação (20.4) para ajustar os parâmetros de um gaussiano a um conjunto de dados. Por outro lado, se os parâmetros de cada componente *fossem conhecidos*, poderíamos, pelo menos em um sentido probabilístico, atribuir cada ponto de dados a um componente. O problema é que não conhecemos nem as atribuições nem os parâmetros.

A ideia básica de EM nesse contexto é *fingir* que conhecemos os parâmetros do modelo e depois deduzir a probabilidade de cada ponto de dados pertencer a cada componente. Depois disso, readaptamos os componentes aos dados, onde cada componente é ajustado ao conjunto de dados inteiro, com cada ponto ponderado pela probabilidade de pertencer a esse componente. O processo iterage até a convergência. Essencialmente, estamos "completando" os dados, deduzindo

distribuições de probabilidades sobre as variáveis ocultas — o componente ao qual pertence cada ponto de dados — com base no modelo atual. Para a mistura de distribuições gaussianas, inicializamos arbitrariamente os parâmetros do modelo de mistura e depois repetimos as duas etapas a seguir:

1. **Etapa E**: Calcular as probabilidades $p_{ij} = P(C= i \mid \mathbf{x}_j)$, a probabilidade de que o dado $\mathbf{x}_j$ tenha sido gerado pelo componente i. Pela regra de Bayes, temos $p_{ij} = \alpha P(\mathbf{x}_j \mid C = i)P(C = i)$. O termo $P(\mathbf{x}_j \mid C = i)$ é simplesmente a probabilidade em $\mathbf{x}_j$ do i-ésimo gaussiano, e o termo $P(C = i)$ é o parâmetro que representa o peso para o i-ésimo gaussiano. Definir $n_i = \Sigma_j\, p_{ij}$.
2. **Etapa M**: Calcular a nova média, a covariância e os pesos de componentes, usando as etapas seguintes em sequência:

$$\begin{aligned}
\boldsymbol{\mu}_i &\leftarrow \sum_j p_{ij}\mathbf{x}_j/n_i \\
\boldsymbol{\Sigma}_i &\leftarrow \sum_j p_{ij}(\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^\top/n_i \\
w_i &\leftarrow n_i/N
\end{aligned}$$

onde N é o número total de pontos de dados. A etapa E,ou etapa de *esperança*, pode ser visualizada como o cálculo dos valores esperados p_{ij} das **variáveis indicadoras** ocultas Z_{ij}, onde Z_{ij} é 1 se o dado $\mathbf{x}_j$ foi gerado pelo i-ésimo componente e 0 em caso contrário. A etapa M, ou etapa de *maximização*, encontra os novos valores dos parâmetros que maximizam a probabilidade logarítmica dos dados, dados os valores esperados das variáveis indicadoras ocultas.

O último modelo que EM aprende quando aplicado aos dados da Figura 20.11(a) é mostrado na Figura 20.11(c); ele é virtualmente indistinguível do modelo original a partir do qual os dados foram gerados. A Figura 20.12(a) representa a probabilidade logarítmica dos dados de acordo com o modelo atual, à medida que EM progride.

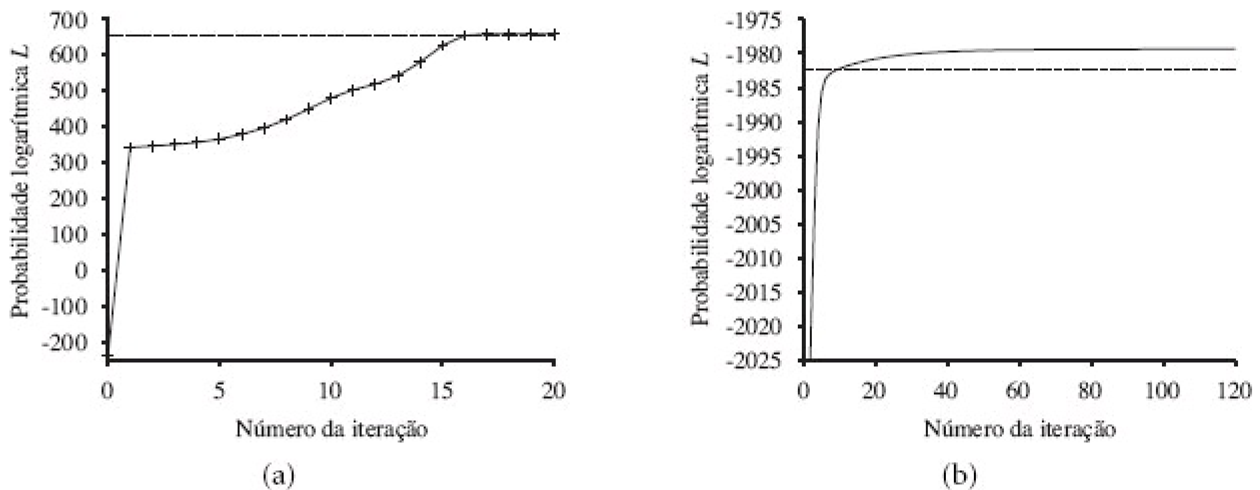


Figura 20.12 Gráficos que mostram a probabilidade logarítmica dos dados, L, como uma função da iteração de EM. A linha horizontal mostra a probabilidade logarítmica de acordo com o modelo verdadeiro. (a) Gráfico correspondente ao modelo de mistura gaussiana da Figura 20.11. (b) Gráfico correspondente à rede bayesiana da Figura 20.13(a).

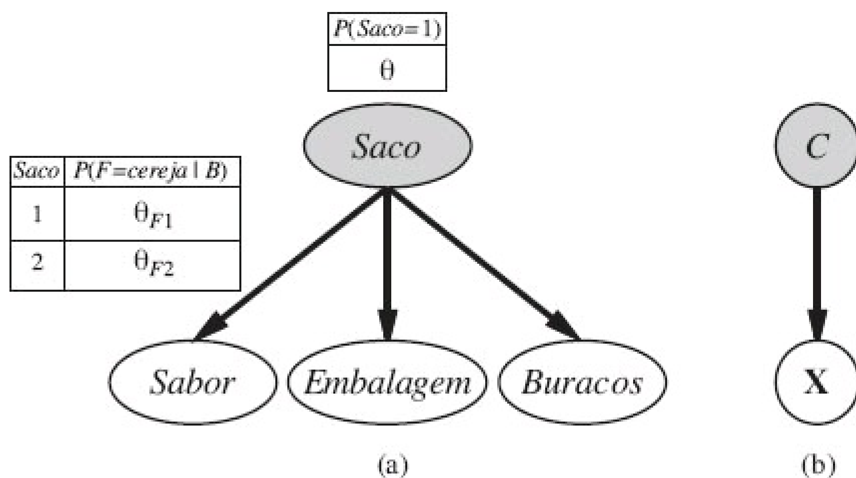


Figura 20.13 (a) Modelo de mistura para doce. As proporções de diferentes sabores, embalagens e presença de buracos dependem do saco, que não é observado. (b) Rede bayesiana para uma mistura gaussiana. A média e a covariância das variáveis observáveis **X** dependem do componente *C*.

Existem dois pontos a serem observados. Primeiro, a probabilidade logarítmica para o modelo aprendido final *excede* ligeiramente a do modelo original, a partir do qual os dados foram gerados. Isso poderia parecer surpreendente, mas, na verdade, simplesmente reflete o fato de que os dados foram gerados ao acaso e não poderiam fornecer um reflexo exato do modelo subjacente. O segundo ponto é que *EM aumenta a probabilidade logarítmica dos dados em cada iteração*. Esse fato pode ser provado no caso geral. Além disso, sob certas condições (válidas na maioria dos casos), pode-se provar que EM alcança um máximo local de probabilidade (em casos raros, ele pode alcançar um ponto de sela ou até um mínimo local). Nesse sentido, EM é semelhante a um algoritmo de subida de encosta baseado em gradiente, mas observe que ele não tem nenhum parâmetro "tamanho do passo".

Nem sempre tudo funciona tão bem quanto a Figura 20.12(a) poderia sugerir. Por exemplo, poderia ocorrer o fato de um componente gaussiano encolher de forma a cobrir apenas um único ponto de dados. Então, sua variância cairá a zero e sua probabilidade tenderá a infinito! Outro problema é que dois componentes podem se "fundir", adquirindo médias e variâncias idênticas e compartilhando seus pontos de dados. Esses tipos de máximos locais degenerados são problemas sérios, em especial no caso de altas dimensões. Uma solução é colocar elementos *a priori* nos parâmetros do modelo e aplicar a versão MAP de EM. Outra é reinicializar um componente com novos parâmetros aleatórios se ele ficar pequeno demais ou próximo demais a outro componente. A inicialização sensível também ajuda.

20.3.2 Aprendizagem de redes bayesianas com variáveis ocultas

Para que uma rede bayesiana com variáveis ocultas possa aprender, aplicamos as mesmas ideias que funcionaram para misturas de gaussianos. A Figura 20.13 representa uma situação em que existem dois sacos de doces que foram misturados. Os doces são descritos por três características: além do *Sabor* e da *Embalagem*, alguns doces têm um *Buraco* no meio e ouros não têm. A distribuição de doces em cada saco é descrita por um modelo **bayesiano ingênuo**: as características são independentes, dado o saco, mas a distribuição de probabilidade condicional para cada característica depende do saco. Os parâmetros são os seguintes: θ é a probabilidade *a priori* de que

um doce venha do *Saco* 1; θ_{F1} e θ_{F2} são as probabilidades de que o sabor seja cereja, dado que o doce vem do *Saco* 1 e do *Saco* 2, respectivamente; θ_{W1} e θ_{W2} fornecem as probabilidades de que a embalagem seja vermelha; e θ_{H1} e θ_{H2} fornecem as probabilidades de que o doce tenha um buraco. Note que o modelo global é um modelo de mistura (de fato, também podemos modelar a mistura de distribuições gaussianas como uma rede bayesiana, como mostra a Figura 20.13(b)). Na figura, o saco é uma variável oculta porque, uma vez que os doces tenham sido misturados, não sabemos mais de qual saco veio cada doce. Em tal caso, podemos recuperar as descrições dos dois sacos observando doces da mistura? Vamos acompanhar uma iteração de EM para esse problema. Primeiro, examinaremos os dados. Geramos 1.000 amostras a partir de um modelo cujos parâmetros verdadeiros são:

$$\theta = 0{,}5,\ \ \theta_{F1} = \theta_{W1} = \theta_{H1} = 0{,}8,\ \ \theta_{F2} = \theta_{W2} = \theta_{H2} = 0{,}3\ . \qquad (20.7)$$

Ou seja, é igualmente provável que os doces tenham vindo de um ou de outro saco; o primeiro contém principalmente doces de cereja com embalagens vermelhas e buracos; o segundo contém principalmente doces de lima com embalagens verdes e nenhum buraco. As contagens para os oito tipos possíveis de doces são dadas a seguir:

	W = *vermelho*		*W* = *verde*	
	H = 1	*H* = 0	*H* = 1	*H* = 0
F = *cereja*	273	93	104	90
F = *lima*	79	100	94	167

Começamos inicializando os parâmetros. Para manter a simplicidade numérica, escolheremos arbitrariamente:[5]

$$\theta^{(0)} = 0{,}6,\ \ \theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6,\ \ \theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0{,}4\ . \qquad (20.8)$$

Primeiro, vamos trabalhar no parâmetro θ. No caso completamente observável, estimaríamos esse parâmetro diretamente a partir das contagens *observadas* de doces dos sacos 1 e 2. Tendo em vista que o saco é uma variável oculta, calculamos em vez disso as contagens *esperadas*. A contagem esperada $\hat{N}$(*Saco* = 1) é a soma, calculada sobre todos os doces da probabilidade de o doce ter vindo do saco 1:

$$\theta^{(1)} = \hat{N}(Saco\ = 1)/N = \sum_{j=1}^{N} P(Saco = 1 \mid Sabor_j,\ embalagem_j,\ buracos_j)/N.$$

Essas probabilidades podem ser calculadas por qualquer algoritmo de inferência para redes bayesianas. Para um modelo bayesiano ingênuo como o de nosso exemplo, podemos fazer a inferência “à mão”, usando a regra de Bayes e aplicando a independência condicional:

$$\theta^{(1)} = \frac{1}{N} \sum_{j=1}^{N} \frac{P(Sabor_j \mid Saco = 1)P(embalagem_j \mid Saco = 1)P(buracos_j \mid Saco = 1)\ P(Saco = 1)}{\sum_i\ P(Sabor_j \mid Saco = i)P(embalagem_j \mid Saco = i)P(buracos_j \mid Saco = i)\ P(Saco = i)}.$$

Aplicando essa fórmula, digamos, aos 273 doces de cereja com embalagens vermelhas e com buracos, conseguimos uma contribuição de:

$$\frac{273}{1000} \cdot \frac{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)}}{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)} + \theta_{F2}^{(0)}\theta_{W2}^{(0)}\theta_{H2}^{(0)}(1-\theta^{(0)})} \approx 0.22797\ .$$

Continuando com os outros sete tipos de doces da tabela de contagens, obtemos $\theta^{(1)} = 0{,}6124$.

Agora vamos considerar os outros parâmetros, como θ_{F1}. No caso completamente observável, estimaríamos isso diretamente a partir das contagens *observadas* de doces de cereja e de lima do saco 1. A contagem *esperada* de doces de cereja provenientes do saco 1 é dada por:

$$\sum_{j:Sabor_j = cereja} P(Saco = 1 \mid Sabor_j = cereja, embalagem_j, buracos_j).$$

Mais uma vez, essas probabilidades podem ser calculadas por qualquer algoritmo de rede bayesiana. Completando esse processo, obtemos os novos valores de todos os parâmetros:

$$\begin{aligned} &\theta^{(1)} = 0{,}6124,\ \theta_{F1}^{(1)} = 0{,}6684,\ \theta_{W1}^{(1)} = 0.6483,\ \theta_{H1}^{(1)} = 0{,}6558, \\ &\theta_{F2}^{(1)} = 0{,}3887, \theta_{W2}^{(1)} = 0{,}3817,\ \theta_{H2}^{(1)} = 0{,}3827\ . \end{aligned} \tag{20.9}$$

A probabilidade logarítmica dos dados aumenta de cerca de –2.044 inicialmente até cerca de –2.021 depois da primeira iteração, como mostra a Figura 20.12(b). Isto é, a atualização melhora a probabilidade propriamente dita por um fator de cerca de $e^{23} \approx 10^{10}$. Pela décima iteração, o modelo aprendido é uma adaptação melhor que o modelo original (L = –1982,214). Depois disso, o progresso se torna muito lento. Isso não é incomum ao usar EM, e muitos sistemas práticos combinam EM com um algoritmo baseado em gradiente como o de Newton–Raphson (veja o Capítulo 4) para a última fase da aprendizagem.

A lição geral a partir desse exemplo é que *as atualizações de parâmetros para aprendizagem de redes bayesianas com variáveis ocultas estão diretamente disponíveis a partir dos resultados de inferência em cada exemplo. Além disso, somente probabilidades posteriores* locais *são necessárias para cada parâmetro*. Aqui, "local" significa que o TPC para cada variável X_i pode ser aprendido de probabilidades posteriores envolvendo apenas X_i e seus pais U_i. Definindo θ_{ijk} como sendo o parâmetro TPC $P(X_i = x_{ij} \mid \mathbf{U}_i = \mathbf{u}_{ik})$, a atualização é dada pelas contagens normalizadas esperadas como a seguir:

$$\theta_{ijk} \leftarrow \hat{N}(X_i = x_{ij}, \mathbf{U}_i = \mathbf{u}_{ik}) / \hat{N}(\mathbf{U}_i = \mathbf{u}_{ik})\ .$$

As contagens esperadas são obtidas efetuando-se o somatório sobre os exemplos, calculando-se as probabilidades $P(X_i = x_{ij} \mid \mathbf{U}_i = \mathbf{u}_{ik})$ para cada uma usando qualquer algoritmo de inferência de rede

bayesiana. Para os algoritmos exatos — inclusive os de eliminação de variáveis —, todas essas probabilidades podem ser obtidas diretamente como um subproduto da inferência-padrão, sem a necessidade de computações extras específicas para aprendizagem. Além disso, as informações necessárias para aprendizagem estão disponíveis *localmente* para cada parâmetro.

20.3.3 Aprendizagem de modelos ocultos de Markov

Nossa aplicação final de EM envolve a aprendizagem das probabilidades de transições em modelos ocultos de Markov (MOMs). Lembre-se de que, na Seção 15.3, vimos que um modelo oculto de Markov pode ser representado por uma rede bayesiana dinâmica com uma única variável de estados discretos, como ilustra a Figura 20.14. Cada ponto de dados consiste em uma *sequência* de observações de duração finita e, assim, o problema é aprender as probabilidades de transições de um conjunto de sequências de observações (ou, possivelmente, a partir de apenas uma sequência longa).

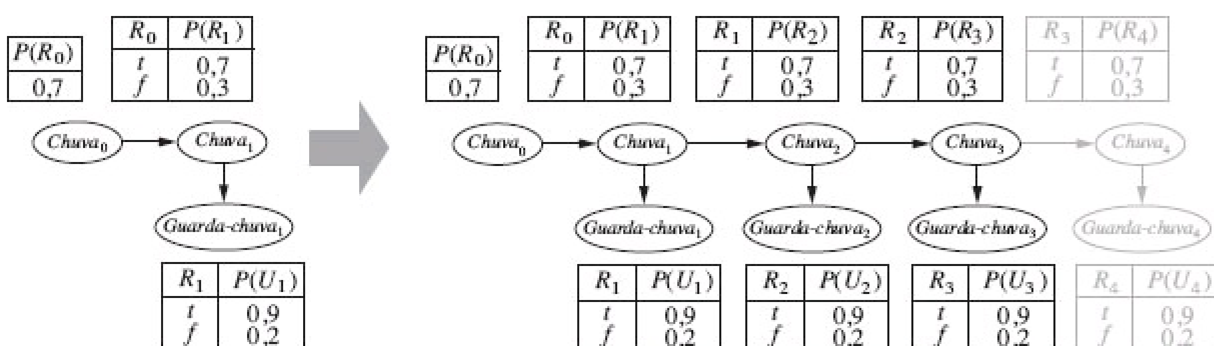


Figura 20.14 Rede bayesiana dinâmica desenrolada que representa um modelo oculto de Markov (repetição da Figura 15.16).

Já descobrimos como fazer aprendizagem em redes bayesianas, mas existe uma complicação: em redes bayesianas, cada parâmetro é distinto; por outro lado, em um modelo oculto de Markov, as probabilidades de transições individuais do estado i para o estado j no tempo t, $\theta_{ijt} = P(X_{t+1} = j \mid X_t = i)$, são *repetidas* ao longo do tempo, isto é, $\theta_{ijt} = \theta_{ij}$ para todo t. Para estimar a probabilidade de transição do estado i para o estado j, simplesmente calculamos a proporção esperada de vezes em que o sistema sofre uma transição para o estado j quando se encontra no estado i:

$$\theta_{ij} \leftarrow \sum_t \hat{N}(X_{t+1} = j, X_t = i) / \sum_t \hat{N}(X_t = i) .$$

Mais uma vez, as contagens esperadas são calculadas por qualquer algoritmo de inferência de MOM. O algoritmo **para a frente-para trás** mostrado na Figura 15.4 pode ser modificado muito facilmente para calcular as probabilidades necessárias. Um ponto importante é que as probabilidades exigidas são aquelas obtidas por **suavização**, em vez de **filtragem**; isto é, precisamos prestar atenção à evidência subsequente na avaliação da probabilidade de ter ocorrido uma transição específica. Como dissemos no Capítulo 15, a evidência em um caso de assassinato normalmente é obtida *depois* de o crime (ou seja, a transição do estado i para o estado j) ter ocorrido.

20.3.4 A forma geral do algoritmo EM

Vimos várias instâncias do algoritmo EM. Cada uma envolve a computação de valores esperados de variáveis ocultas para cada exemplo, e depois a repetição dos cálculos dos parâmetros usando os valores esperados como se eles fossem valores observados. Seja **x** o conjunto de todos os valores observados em todos os exemplos, seja **Z** o conjunto de todas as variáveis ocultas para todos os exemplos, e seja q o conjunto de todos os parâmetros para o modelo de probabilidade. Então, o algoritmo EM é:

$$\boldsymbol{\theta}^{(i+1)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{\mathbf{z}} P(\mathbf{Z}=\mathbf{z} \mid \mathbf{x}, \boldsymbol{\theta}^{(i)}) L(\mathbf{x}, \mathbf{Z}=\mathbf{z} \mid \boldsymbol{\theta}) .$$

Essa equação é o algoritmo EM em resumo. A etapa E é o cálculo do somatório, que corresponde à esperança da probabilidade logarítmica dos dados “completados” com relação à distribuição $P(\mathbf{Z} = \mathbf{z}|\mathbf{x}, \theta(i))$, que é a posterior sobre as variáveis ocultas, considerando-se os dados. A etapa M é a maximização dessa probabilidade logarítmica esperada com relação aos parâmetros. Para misturas de gaussianos, as variáveis ocultas são os valores Z_{ij}, onde Z_{ij} é 1 se o exemplo j foi gerado pelo componente i. Para redes bayesianas, Z_{ij} é o valor da variável não observada X_i no exemplo j. Para MOMs, Z_{ij} é o estado da sequência no exemplo j no tempo t. A partir da forma geral, é possível derivar um algoritmo EM para uma aplicação específica, uma vez que tenham sido identificadas as variáveis ocultas apropriadas.

Tão logo compreendemos a ideia geral de EM, fica fácil derivar todas as espécies de variantes e aperfeiçoamentos. Por exemplo, em muitos casos a etapa E — a computação de posteriores sobre as variáveis ocultas — é intratável, como em grandes redes bayesianas. Ocorre que é possível usar uma etapa E *aproximada*, e ainda assim obter um algoritmo de aprendizagem efetivo. Com um algoritmo de amostragem como o CMMC (veja a Seção 14.5), o processo de aprendizagem é muito intuitivo: cada estado (configuração de variáveis ocultas e observadas) visitado por CMMC é tratada exatamente como se fosse uma observação completa. Desse modo, os parâmetros podem ser atualizados diretamente depois de cada transição de CMMC. Outras formas de inferência aproximada, como métodos variacionais e iterativos, também se mostraram efetivos para a aprendizagem de redes muito grandes.

20.3.5 Aprendizagem de estruturas de redes bayesianas com variáveis ocultas

Na Seção 20.2.5, discutimos o problema de aprendizagem em estruturas de redes bayesianas com dados completos. Quando as variáveis não observadas podem influenciar os dados que são observados, tudo fica mais difícil. No caso mais simples, um especialista humano pode informar ao algoritmo de aprendizagem que certas variáveis ocultas existem, deixando para o algoritmo encontrar um lugar para elas na estrutura da rede. Por exemplo, um algoritmo poderia tentar aprender a estrutura mostrada na Figura 20.10(a) dada a informação de que *DoençaDoCoração* (uma variável de três valores) deve ser incluída no modelo. Como no caso de dados completos, o algoritmo tem um laço externo que busca sobre as estruturas e um laço interno que se encaixa nos parâmetros da rede,

dada a estrutura.

Se o algoritmo de aprendizagem não receber a informação de que as variáveis ocultas existem, haverá duas escolhas: fingir que os dados estão realmente completos — o que pode forçar o algoritmo a aprender o modelo de parâmetros intensivos da Figura 20.10(b) — ou *criar* novas variáveis ocultas, a fim de simplificar o modelo. Esta última abordagem pode ser implementada pela inclusão de novas opções de modificação de estrutura na busca: além de modificar vínculos, o algoritmo pode adicionar ou eliminar uma variável oculta, ou mudar sua aridade. É claro que o algoritmo não saberá que a nova variável que criou é chamada *DoençaDoCoração*; nem terá nomes significativos para os valores. Felizmente, variáveis ocultas recém-criadas em geral estarão conectadas a variáveis preexistentes e, assim, um especialista humano poderá com frequência inspecionar as distribuições condicionais locais que envolvem a nova variável e averiguar seu significado.

Como no caso de dados completos, a aprendizagem da estrutura pura de máxima probabilidade resultará em uma rede completamente conectada (além disso, uma rede sem variáveis ocultas) e, assim, é necessária alguma forma de penalidade para a complexidade. Também podemos aplicar CMMC para realizar a amostragem de muitas estruturas de rede possíveis, desse modo se aproximando à aprendizagem bayesiana. Por exemplo, podemos aprender misturas de gaussianos com um número desconhecido de componentes, realizando a amostragem sobre o número; a distribuição posterior aproximada para o número de gaussianos é dada pelas frequências de amostragem do processo CMMC.

Para o caso de dados completos, o laço interno para ser informado dos parâmetros é muito rápido — somente uma questão de extrair frequências condicionais do conjunto de dados. Quando houver variáveis ocultas, o laço interno poderá envolver muitas iterações de EM ou um algoritmo baseado em gradiente, e cada iteração envolverá o cálculo de posteriores em uma rede bayesiana, que é por si só um problema NP-difícil. Até agora, essa abordagem provou ser impraticável para a aprendizagem de modelos complexos. Um aperfeiçoamento possível é o algoritmo chamado **EM estrutural**, que opera de maneira quase idêntica ao algoritmo EM comum (paramétrico), exceto pelo fato de que o algoritmo pode atualizar a estrutura e também os parâmetros. Da mesma maneira como o EM comum utiliza os parâmetros atuais para calcular as contagens esperadas na etapa E, e depois aplica essas contagens na etapa M para escolher novos parâmetros, o EM estrutural emprega a estrutura atual para calcular as contagens esperadas e depois aplica essas contagens à etapa M para avaliar a probabilidade de novas estruturas potenciais (isso contrasta com o método de laço externo/laço interno, que calcula novas contagens esperadas para cada estrutura potencial). Desse modo, o EM estrutural pode fazer várias alterações estruturais para a rede sem recalcular as contagens esperadas, e é capaz de aprender estruturas de redes bayesianas não triviais. Todavia, ainda resta muito trabalho a ser feito antes de podermos afirmar que o problema de aprendizagem de estrutura foi resolvido.

20.4 RESUMO

Os métodos estatísticos de aprendizagem variam desde o cálculo simples de médias até a construção de modelos complexos, como redes bayesianas e redes neurais. Eles têm aplicações ao

longo de toda a ciência da computação em engenharia, biologia computacional, neurociência, psicologia e física. Este capítulo apresentou algumas das ideias básicas e forneceu uma visão dos conceitos matemáticos. Os pontos principais são:

- Os métodos de **aprendizagem bayesiana** formulam a aprendizagem como uma forma de inferência probabilística usando as observações para atualizar uma distribuição *a priori* sobre hipóteses. Essa abordagem fornece um bom caminho para implementar a lâmina de Ockham, mas logo se torna intratável para espaços de hipóteses complexos.
- A aprendizagem de **máximo *a posteriori*** (MAP) seleciona uma única hipótese mais provável, considerando-se os dados. A hipótese *a priori* ainda é usada, e o método frequentemente é mais tratável que a aprendizagem bayesiana total.
- A aprendizagem de **máxima probabilidade** simplesmente seleciona a hipótese que maximiza a probabilidade dos dados; ela é equivalente à aprendizagem de MAP com um *a priori* uniforme. Em casos simples, como a regressão linear e as redes bayesianas completamente observáveis, as soluções de máxima probabilidade podem ser encontradas com facilidade em forma fechada. A aprendizagem bayesiana **ingênua** é uma técnica particularmente efetiva que se ajusta bem à escala.
- Quando algumas variáveis são ocultas, podem ser encontradas soluções de máxima probabilidade local com o uso do algoritmo EM. As aplicações incluem formação de agrupamentos com a utilização de misturas de gaussianos, aprendizagem de redes bayesianas e aprendizagem de modelos ocultos de Markov.
- A aprendizagem da estrutura de redes bayesianas é um exemplo de **seleção de modelos**. Em geral, essa técnica envolve uma busca discreta no espaço de estruturas. É necessário algum método para tratar o compromisso entre a complexidade do modelo e o grau de adaptação.
- **Modelos não paramétricos** representam uma distribuição que utiliza a coleção de pontos de dados. Desse modo, o número de parâmetros cresce com o conjunto de treinamento. Os métodos de **vizinho mais próximo** examinam as instâncias mais próximas ao ponto em questão, enquanto os métodos de **kernel** formam uma combinação com ponderação de distâncias de todos os exemplos.

A aprendizagem estatística continua a ser uma área muito ativa de pesquisa. Enormes avanços foram feitos, tanto na teoria quanto na prática, até o ponto em que é possível ter a aprendizagem de quase qualquer modelo para o qual seja viável a inferência exata ou aproximada.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

A aplicação de técnicas de aprendizagem estatística em IA foi uma área ativa de pesquisa nos primeiros anos (veja Duda e Hart, 1973), mas se separou da corrente principal da IA à medida que este último campo se concentrou em métodos simbólicos. Um ressurgimento do interesse ocorreu logo depois da introdução de modelos de redes bayesianas no final da década de 1980; aproximadamente na mesma época, começou a emergir uma visão estatística da aprendizagem de redes neurais. No final da década de 1990, havia uma notável convergência de interesses em

aprendizagem de máquina, estatística e redes neurais, concentrada em métodos para criação de grandes modelos probabilísticos a partir dos dados.

O modelo bayesiano ingênuo é uma das mais antigas e mais simples formas de rede bayesiana, datando da década de 1950. Suas origens foram mencionadas nas notas do final do Capítulo 13. Ele foi parcialmente explicado por Domingos e Pazzani (1997). Uma forma reforçada de aprendizagem bayesiana ingênua ganhou a primeira competição de mineração de dados KDD Cup (Elkan, 1997). Heckerman (1998) apresenta uma excelente introdução ao problema geral de aprendizagem de redes bayesianas. A aprendizagem de parâmetros bayesiana com prioris de Dirichlet para redes bayesianas foi discutida por Spiegelhalter *et al*. (1993). O pacote de software BUGS (Gilks *et al*., 1994) incorpora muitas dessas ideias e fornece uma ferramenta muito poderosa para formular e aprender modelos de probabilidade complexos. Os primeiros algoritmos para aprendizagem de estruturas de rede bayesiana utilizavam testes de independência condicional (Pearl, 1988; Pearl e Verma, 1991). Spirtes *et al.* (1993) desenvolveram uma abordagem completa incorporada com o pacote TETRAD para aprendizagem de rede bayesiana, utilizando ideias semelhantes. Os aperfeiçoamentos algorítmicos ocorridos desde então levaram a uma clara vitória na competição de mineração de dados KDD Cup de 2001 de um método de aprendizagem de rede bayesiana (Cheng *et al*., 2002). (Nesse caso, a tarefa específica foi um problema de bioinformática com 139.351 características!) Uma abordagem de aprendizagem de estruturas baseada na maximização da probabilidade foi desenvolvida por Cooper e Herskovits (1992) e otimizada por Heckerman *et al.* (1994). Desde aquela época, vários avanços em algoritmos levaram a um desempenho bastante respeitável no caso dos dados completos (Moore e Wong, 2003; Teyssier e Koller, 2005). Um componente importante é uma estrutura de dados eficiente, a árvore AD, que conta com todas as combinações de variáveis e valores possíveis em cache (Moore e Lee, 1997). Friedman e Goldszmidt (1996) assinalaram a influência da representação de distribuições condicionais locais sobre a estrutura aprendida.

O problema geral de aprendizagem de modelos de probabilidade com variáveis ocultas e dados omitidos foi tratado por Hartley (1958), que descreveu a ideia geral do que foi mais tarde foi chamado EM e deu vários exemplos. Um impulso maior veio do algoritmo de Baum−Welch, para aprendizagem de MOM (Baum e Petrie, 1966), que é um caso especial de EM. A tese de Dempster, Laird e Rubin (1977), que apresentaram o algoritmo EM na forma geral e analisaram sua convergência, é um dos trabalhos mais citados na ciência da computação e estatística. (O próprio Dempster vê o EM como um esquema, e não como um algoritmo, pois pode ser necessária grande dose de elaboração matemática antes que ele possa ser aplicado a uma nova família de distribuições.) McLachlan e Krishnan (1997) dedicaram um livro inteiro ao algoritmo e suas propriedades. O problema específico de aprendizagem de modelos de misturas, inclusive misturas de gaussianos, é focalizado por Titterington *et al.* (1985). Dentro da IA, o primeiro sistema bem-sucedido que utilizou o EM para modelagem de mistura foi o AUTOCLASS (Cheeseman *et al*., 1988; Cheeseman e Stutz, 1996). O AUTOCLASS foi aplicado a uma série de tarefas reais de classificação científica, incluindo a descoberta de novos tipos de estrelas a partir de dados espectrais (Goebel *et al*., 1989) e novas classes de proteínas e íntrons em bancos de dados de sequências de DNA/proteínas (Hunter e States, 1992).

Para o parâmetro de aprendizagem de probabilidade máxima em redes de Bayes com variáveis ocultas, EM e métodos baseados em gradiente foram introduzidos ao mesmo tempo por Lauritzen

(1995), Russell *et al.* (1995) e Binder *et al*. (1997a). O algoritmo EM estrutural foi desenvolvido por Friedman (1998) e aplicado para a probabilidade máxima de estrutura de aprendizagem de redes bayesianas com variáveis latentes. Friedman e Koller (2003) descreveram a estrutura de aprendizagem bayesiana.

A habilidade de aprender a estrutura de redes bayesianas está intimamente relacionada à questão de recuperar informações *causais* a partir dos dados. Ou seja, é possível aprender redes bayesianas de tal modo que a estrutura de rede recuperada indique influências causais reais? Por muitos anos, os estatísticos evitaram essa questão, acreditando que dados de observações (em oposição a dados gerados a partir de testes experimentais) poderiam gerar apenas informações correlacionadas — afinal, duas variáveis quaisquer que parecem inter-relacionadas poderiam de fato ser influenciadas por um terceiro fator causal desconhecido, em vez de influenciarem diretamente uma à outra. Pearl (2000) apresentou argumentos convincentes da ideia oposta, mostrando que existem de fato muitos casos em que a causalidade pode ser averiguada, e também desenvolvendo o formalismo de **rede causal** para expressar as causas e os efeitos da intervenção, bem como probabilidades condicionais comuns.

A estimação da densidade não paramétrica, também chamada estimativa de densidade de **janela de Parzen**, foi investigada inicialmente por Rosenblatt (1956) e Parzen (1962). Desde aquela época, foi desenvolvida vasta literatura sobre a investigação das propriedades de vários avaliadores. Devroye (1987) apresenta uma introdução completa.

Há também uma literatura crescente sobre métodos bayesianos não paramétricos, originados com o trabalho seminal de Ferguson (1973) sobre o **processo de Dirichlet**, que pode ser considerado como uma distribuição sobre as distribuições de Dirichlet. Esses métodos são particularmente úteis para misturar com números desconhecidos de componentes. Ghahramani (2005) e Jordan (2005) oferecem tutoriais úteis sobre as muitas aplicações dessas ideias para aprendizagem de estatística. O texto de Rasmussen e Williams (2006) aborda o **processo gaussiano**, que apresenta uma maneira de definir as distribuições anteriores sobre o espaço das funções contínuas.

O material deste capítulo reúne o trabalho de pesquisa nos campos de estatística e reconhecimento de padrões e redes neurais, e, assim, o assunto foi repetido muitas vezes de várias maneiras. Bons textos em estatísticas de Bayes incluem os de DeGroot (1970), Berger (1985) e Gelman *et al.* (1995). Bishop (2007) e Hastie *et al.* (2001) fornecem uma excelente introdução aos métodos estatísticos de aprendizagem. Para classificação de padrões, o texto clássico durante muitos anos foi o de Duda e Hart (1973), agora atualizado (Duda *et al*., 2001). A conferência anual NIPS (Neural Information Processing Conference), cujos anais são publicados como a série *Advances in Neural Information Processing Systems,* agora é dominadas por artigos bayesianos. Também aparecem documentos sobre aprendizagem de redes bayesianas nas conferências *Uncertainty in AI* e *Machine Learning* e em diversas conferências de estatística. Periódicos específicos para redes neurais incluem *Neural Computation*, *Neural Networks* e *IEEE Transactions on Neural Networks*. Reuniões bayesianas especificamente incluem o Valencia International Meetings on Bayesian Statics e a revista *Bayesian Analysis*.

EXERCÍCIOS

20.1 Os dados usados para a Figura 20.1 na página 804 podem ser visualizados como sendo gerados por h_5. Para cada uma das outras quatro hipóteses, gere um conjunto de dados de comprimento 100 e represente os grafos correspondentes para $P(h_i \mid d_1,\ldots, d_n)$ e $P(D_{n+1} = lima \mid d_1,\ldots, d_n)$. Comente seus resultados.

20.2 Suponha que as utilidades de Ana para doces de cereja e lima sejam c_A e l_A, enquanto as utilidades de Bob são c_B e l_B (porém, uma vez que Ana tiver desembrulhado um pedaço de doce, Bob não o comprará). Presumivelmente, se Bob gostar de doces de lima muito mais que Ana, será sensato para Ana vender seu saco de doces, já que ela tem certeza de que o saco contém doces de lima. Por outro lado, se Ana desembrulhar muitos doces no processo, o saco valerá menos. Discuta o problema de determinar o ponto ótimo para venda do saco. Determine a utilidade esperada do procedimento ótimo, dada a distribuição *a priori* da Seção 20.1.

20.3 Dois estatísticos vão ao médico e ambos recebem o mesmo prognóstico: uma chance de 40% de que o problema seja a doença mortal *A* e 60% de chance de ter a doença fatal *B*. Felizmente, existem no mercado remédios anti-*A* e anti-*B* que são econômicos, 100% eficazes e livres de efeitos colaterais. Os estatísticos têm a opção de usar um dos remédios, ambos ou nenhum dos dois. O que fará o primeiro estatístico (um adepto de Bayes)? E quanto ao segundo estatístico, que sempre utiliza a hipótese de máxima probabilidade?

O médico realiza alguma pesquisa e descobre que, na realidade, existem duas versões da doença *B*, dextro-*B* e levo-B, que são igualmente prováveis e têm igual possibilidade de tratamento pelo remédio anti-*B*. Agora, que existem três hipóteses, o que os dois estatísticos farão?

20.4 Explique como aplicar o método de aceleração do Capítulo 18 à aprendizagem bayesiana ingênua. Teste o desempenho do algoritmo resultante no problema de aprendizagem de restaurante.

20.5 Considere *N* pontos de dados (x_j, y_j), onde os valores y_j são gerados a partir dos valores x_j de acordo com o modelo gaussiano linear da Equação 20.5. Encontre os valores de θ_1, θ_2 e s que maximizam a probabilidade logarítmica condicional dos dados.

20.6 Considere o modelo de OU-ruidoso para febre descrito na Seção 14.3. Explique como aplicar a aprendizagem de máxima probabilidade para ajustar os parâmetros de tal modelo a um conjunto de dados completos. (*Sugestão*: Utilize a regra do encadeamento para derivar as parciais.)

20.7 Este exercício investiga as propriedades da distribuição beta definida na Equação 20.6.

a. Por integração sobre o intervalo [0, 1], mostre que a normalização permanente para a distribuição beta$[a, b]$ é dada por a $= \Gamma(a + b)/\Gamma(a)\Gamma(b)$, onde $\Gamma(x)$ é a **função gama**, definida por $\Gamma(x + 1) = x. \Gamma(x)$ e por $\Gamma(1) = 1$. (Para o inteiro x, $\Gamma(x + 1) = x!$.)

b. Mostre que a média é $a/(a + b)$.

c. Encontre a(s) moda(s) (o(s) valor(es) mais provável(is) de θ).

d. Descreva a distribuição beta$[\epsilon, \epsilon]$ para ϵ muito pequeno. O que acontece à medida que tal distribuição é atualizada?

20.8 Considere uma rede bayesiana arbitrária, um conjunto de dados completos para essa rede e a probabilidade para o conjunto de dados de acordo com a rede. Forneça uma prova simples de que a

probabilidade dos dados não pode diminuir se adicionarmos um novo vínculo à rede e recalcularmos os valores de parâmetros de máxima probabilidade.

20.9 Considere uma variável Y aleatória booleana única (a "classificação"). Faça com que a primeira prioridade $P(Y = verdadeiro)$ seja π. Tentemos encontrar π dado o conjunto de treinamento $(D = (y_1,\ldots,y_n)$ com N amostras independentes de Y. Além disso, suponha que p de N seja positivo e n de N seja negativo.

a. Escreva uma expressão para a probabilidade de D (ou seja, a probabilidade de ver essa sequência particular de exemplos, dado um valor fixo de π em termos de π, p e n).

b. Diferenciando a probabilidade logarítmica L, encontre o valor de p que maximiza a probabilidade.

c. Suponha agora que adicionemos k variáveis aleatórias booleanas $X_1, X_2,\ldots X_k$ (os "atributos") que descrevem cada amostra e suponha que se assuma que esses atributos sejam condicionais, independentes uns dos outros dado o objetivo Y. Desenhe a rede bayesiana correspondente desse pressuposto.

d. Escreva a probabilidade dos dados incluindo os atributos, utilizando a seguinte notação adicional:

- α_i é $P(X_i = verdadeiro \mid Y = verdadeiro)$.
- β_i é $P(X_i = verdadeiro \mid Y = falso)$
- p_i^+ é a contagem de amostras para a qual $X_i = verdadeiro$ e $Y = verdadeiro$.
- n_i^+ é a contagem de amostras para a qual $X_i = falso$ e $Y = verdadeiro$.
- p_i^- é a contagem de amostras para a qual $X_i = verdadeiro$ e $Y = falso$.
- n_i^- é a contagem de amostras para a qual $X_i = falso$ e $Y = falso$.

[*Dica:* considere a primeira probabilidade de ver um único exemplo com os valores especificados para $X_1, X_2,\ldots X_k$ e Y.]

e. Diferenciando a probabilidade logarítmica L, encontre os valores α_1 e θ_i (em termos de várias contagens) que maximize a probabilidade e diga em palavras o que esses valores representam.

f. Faça $k = 2$ e considere um conjunto de dados com todos os quatro exemplos possíveis da função XOR. Calcule a estimativa de probabilidade máxima de π, α_1, α_2, β_1 e β_2.

g. Dadas as estimativas de π, α_1, α_2, β_1 e β_2 qual é a probabilidade posterior $P(Y = verdadeiro|x_1, x_2)$ de cada exemplo?

20.10 Considere a aplicação de EM à aprendizagem dos parâmetros para a rede na Figura 20.13(a), dados os parâmetros verdadeiros da Equação 20.7.

a. Explique por que o algoritmo EM não funcionaria se existissem apenas dois atributos no modelo, em vez de três.

b. Mostre os cálculos para a primeira iteração de EM a partir da Equação 20.8.

c. O que acontece se iniciarmos com todos os parâmetros definidos com o mesmo valor p?

(*Sugestão*: Talvez você considere útil investigar essa questão empiricamente antes de derivar o resultado geral.)

d. Escreva uma expressão para a probabilidade logarítmica dos dados de doces tabulados na página 821 em termos dos parâmetros, calcule as derivadas parciais com relação a cada parâmetro e investigue a naturezado ponto fixo alcançado na parte (c).

---

[1] Os leitores com bons conhecimentos de estatística reconhecerão esse cenário como uma variante do problema de **urna e bola**. Consideramos urnas e bolas menos interessantes que doces; além disso, doces também são apropriados para outras tarefas como, por exemplo, decidir se devemos ou não negociar o saco de doces com um amigo — veja o Exercício 20.2.

[2] Declaramos antes que os sacos de doces são muito grandes; caso contrário, a suposição de i.i.d. deixa de ser válida. Tecnicamente, é mais correto (embora menos higiênico) reembalar cada doce após a inspeção e devolvê-lo ao saco.

[3] Eles são chamados hiperparâmetros porque parametrizam uma distribuição sobre θ, que é ele próprio um parâmetro.

[4] Outros conjugados *a priori* incluem a família **Dirichlet** para os parâmetros de uma distribuição multivalorada discreta e a família **Normal-Wishart** para os parâmetros de uma distribuição gaussiana. Veja Bernardo e Smith (1994).

[5] Na prática, é melhor escolher esses valores aleatoriamente, a fim de evitar máximos locais devidos à simetria.

CAPÍTULO 21

Aprendizagem por reforço

Em que examinamos como um agente pode aprender a partir do sucesso e do fracasso, da recompensa e penalidade.

21.1 INTRODUÇÃO

Os Capítulos 18, 19 e 20 abordaram métodos de aprendizagem que aprendem funções, teorias lógicas e modelos de probabilidade a partir de exemplos. Neste capítulo, estudaremos como os agentes podem aprender *o que fazer*, na ausência de exemplos rotulados.

Considere, por exemplo, o problema de aprender a jogar xadrez. Um agente de aprendizagem supervisionada precisa ser informado da jogada correta para cada posição que encontra, mas tal realimentação raramente está disponível. Na ausência da realimentação de um professor, um agente pode aprender um modelo de transição para seus próprios movimentos e talvez possa aprender a prever as jogadas do adversário, mas *sem alguma realimentação sobre o que é bom e o que é ruim*, *o agente não terá nenhuma base para decidir que movimento executar*. O agente precisa saber que algo de bom aconteceu quando (acidentalmente) dá o xeque-mate no oponente ou vice versa, se for um jogo de xadrez suicida. Essa espécie de realimentação é chamada **recompensa** ou **reforço**. Em jogos como o xadrez, o reforço é recebido apenas no fim do jogo. Em outros ambientes, as recompensas vêm com maior frequência. No jogo de pingue-pongue, cada ponto marcado pode ser considerado uma recompensa; quando se aprende a engatinhar, qualquer movimento para a frente é uma realização. Nossa estrutura para agentes considera a recompensa como uma *parte* da percepção de entrada, mas o agente deve ser "fisicamente programado" para reconhecer essa parte como uma recompensa, e não apenas como outra entrada sensória. Desse modo, os animais parecem estar programados para reconhecer dor e fome como recompensas negativas e também prazer e ingestão de alimentos como recompensas positivas. O reforço foi cuidadosamente investigado por estudiosos da psicologia animal por mais de 60 anos.

As recompensas foram introduzidas no Capítulo 17, onde serviram para definir políticas ótimas em **processos de decisão de Markov** (MDPs). Uma política ótima é uma política que maximiza a recompensa total esperada. A tarefa da **aprendizagem por reforço** consiste em usar recompensas observadas para aprender uma política ótima (ou quase ótima) para o ambiente. Enquanto no Capítulo 17 o agente tinha um modelo completo do ambiente e conhecia a função de recompensa,

aqui não supomos nenhum conhecimento anterior do modelo ou da função de recompensa. Imagine disputar um novo jogo cujas regras você não conhece; depois de aproximadamente uma centena de movimentos, seu oponente anuncia: "Você perdeu." Em resumo, isso é a aprendizagem por reforço.

Em muitos domínios complexos, a aprendizagem por reforço é o único caminho possível para treinar um programa com desempenho de alto nível. Por exemplo, em jogos, é muito difícil um ser humano fornecer avaliações precisas e consistentes de um grande número de posições, que seriam necessárias para treinar uma função de avaliação diretamente a partir de exemplos. Em vez disso, o programa pode ser informado de quando ganhou ou perdeu, e pode usar essa informação para aprender uma função de avaliação que forneça estimativas razoavelmente precisas da probabilidade de ganhar a partir de qualquer posição dada. De modo semelhante, é extremamente difícil programar um agente para voar em um helicóptero; ainda assim, dadas recompensas negativas por cair, colidir ou se desviar de um curso definido, um agente poderá aprender a voar por si só.

Pode-se considerar que a aprendizagem por reforço abrange toda a IA: um agente é colocado em um ambiente e tem de aprender a se comportar com sucesso nesse ambiente. Para manter o capítulo tratável, vamos nos concentrar em ambientes simples e projetos de agentes simples. Na maior parte do capítulo, vamos supor um ambiente completamente observável, de forma que o estado atual seja fornecido por cada percepção.

Por outro lado, vamos supor que o agente não sabe como o ambiente funciona ou que ações executar, e permitiremos resultados de ações probabilísticas. Portanto, o agente interaje com um MDP. Vamos considerar três dos projetos de agentes introduzidos pela primeira vez no Capítulo 2:

- Um **agente baseado na utilidade** aprende uma função utilidade sobre estados e a utiliza para selecionar as ações que maximizem a utilidade esperada do resultado.
- Um agente de **aprendizagem Q** (Q-learning) aprende uma função **ação-valor**, ou função *Q*, que fornece a utilidade esperada de se adotar uma dada ação em um estado específico.
- Um **agente reativo** aprende uma política que faz o mapeamento direto de estados para ações.

Um agente baseado na utilidade também deve ter um modelo do ambiente, a fim de tomar decisões porque ele deve conhecer os estados aos quais suas ações levarão. Por exemplo, para fazer uso de uma função de avaliação de gamão, um programa de gamão deve saber quais são seus movimentos válidos *e como eles afetam a posição no tabuleiro*. Somente desse modo ele poderá aplicar a função utilidade aos estados resultantes. Por outro lado, um agente de aprendizagem *Q* pode comparar as utilidades esperadas de suas escolhas disponíveis sem precisar conhecer seus resultados e, assim, não precisa de um modelo do ambiente. Em contrapartida, por não saberem aonde suas ações levam, os agentes de aprendizagem *Q* não podem realizar uma observação antecipada; isso pode restringir seriamente sua habilidade para aprender, como veremos.

Começaremos na Seção 21.2 com a **aprendizagem passiva**, em que a política do agente é fixa e a tarefa consiste em aprender as utilidades de estados (ou pares estado-ação); isso também poderia envolver a aprendizagem de um modelo do ambiente. A Seção 21.3 estuda a **aprendizagem ativa**, em que o agente também deve aprender o que fazer. A principal questão é a **exploração**: um agente deve experimentar tanto quanto possível do seu ambiente, a fim de aprender como se comportar nele. A Seção 21.4 examina como um agente pode utilizar a aprendizagem indutiva para aprender muito

mais rápido a partir de suas experiências. A Seção 21.5 cobre métodos para aprendizagem de representações diretas de políticas em agentes reativos. A compreensão dos processos de decisão de Markov (Capítulo 17) é essencial para este capítulo.

21.2 APRENDIZAGEM POR REFORÇO PASSIVA

Para manter a simplicidade, começaremos com o caso de um agente de aprendizagem passiva que utiliza uma representação baseada em estados em um ambiente completamente observável. Na aprendizagem passiva, a política π do agente é fixa: no estado s, ele sempre executa a ação $\pi(s)$. Sua meta é simplesmente aprender o quanto a política é boa, ou seja, aprender a função utilidade $U^{\pi}(s)$. Usaremos como nosso exemplo o mundo 4 × 3 introduzido no Capítulo 17. A Figura 21.1 mostra uma política para esse mundo e as utilidades correspondentes. É claro que a tarefa de aprendizagem passiva é semelhante à tarefa de **avaliação de política**, uma parte do algoritmo de **iteração de política** descrito na Seção 17.3. A principal diferença é que o agente de aprendizagem passiva não conhece o **modelo de transição** $P(s' \mid s, a)$, que especifica a probabilidade de alcançar o estado s' a partir do estado s depois de realizar a ação a; ele também não conhece a **função recompensa** $R(s)$, que especifica a recompensa para cada estado.

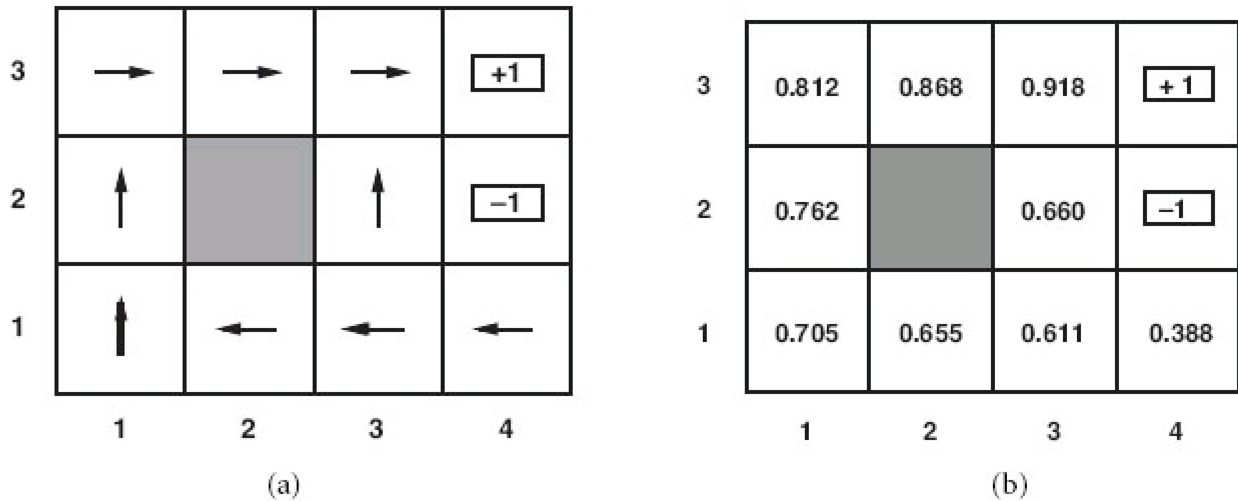


Figura 21.1 (a) Uma política π para o mundo 4 × 3; essa política é ótima com recompensas iguais a $R(s) = -0{,}04$ nos estados não terminais e sem desconto. (b) Utilidades dos estados no mundo 4 × 3, dada a política π.

O agente executa um conjunto de **experiências** no ambiente usando sua política π. Em cada experiência, o agente começa no estado (1,1) e experimenta uma sequência de transições de estados até alcançar um dos estados terminais, (4,2) ou (4,3). Suas percepções fornecem tanto o estado atual quanto a recompensa recebida nesse estado. Experimentos típicos seriam:

$$(1,1)_{-0,04} \rightsquigarrow (1,2)_{-0,04} \rightsquigarrow (1,3)_{-0,04} \rightsquigarrow (1,2)_{-0,04} \rightsquigarrow (1,3)_{-0,04} \rightsquigarrow (2,3)_{-0,04} \rightsquigarrow (3,3)_{-0,04} \rightsquigarrow (4,3)_{+1}$$
$$(1,1)_{-0,04} \rightsquigarrow (1,2)_{-0,04} \rightsquigarrow (1,3)_{-0,04} \rightsquigarrow (2,3)_{-0,04} \rightsquigarrow (3,3)_{-0,04} \rightsquigarrow (3,2)_{-0,04} \rightsquigarrow (3,3)_{-0,04} \rightsquigarrow (4,3)_{+1}$$
$$(1,1)_{-0,04} \rightsquigarrow (2,1)_{-0,04} \rightsquigarrow (3,1)_{-0,04} \rightsquigarrow (3,2)_{-0,04} \rightsquigarrow (4,2)_{-1}.$$

Observe que cada percepção de estado tem como subscrito a recompensa recebida. O objetivo é utilizar as informações sobre recompensas para aprender a utilidade esperada $U^{\pi}(s)$ associada a

cada estado não terminal s. A utilidade é definida como a soma esperada de recompensas (descontadas) obtidas se a política π é seguida. Como na Equação 17.2, escrevemos

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right] \tag{21.1}$$

onde $R(s)$ é a recompensa para o estado, S_t (uma variável aleatória) é o estado alcançado no tempo t quando é executada a política π e $S_0 = s$. Incluiremos um **fator de desconto** γ em todas as nossas equações, mas, para o mundo 4 × 3, definiremos γ = 1.

21.2.1 Estimativa de utilidade direta

Um método simples para **estimativa de utilidade direta** foi criado no final da década de 1950, na área de **teoria de controle adaptativo** por Widrow e Hoff (1960). A ideia é que a utilidade de um estado é a recompensa total esperada a partir desse estado em diante (chamado de **recompensa a obter**), e cada teste fornece uma *amostra* dessa quantidade para cada estado visitado. Por exemplo, a primeira das três sequências de experiências dadas anteriormente fornece uma recompensa total amostrada de 0,72 para o estado (1,1), duas amostras de 0,76 e 0,84 para (1,2), duas amostras de 0,80 e 0,88 para (1,3), e assim por diante. Desse modo, no final de cada sequência, o algoritmo calcula a recompensa observada daí em diante para cada estado e atualiza a utilidade estimada correspondente a esse estado, simplesmente mantendo uma média atual para cada estado em uma tabela. No limite de um número infinitamente grande de experiências (trials), a amostra média convergirá para a verdadeira expectativa da Equação 21.1.

É claro que a estimativa de utilidade direta é apenas uma instância da aprendizagem supervisionada, onde cada exemplo tem o estado como entrada e a recompensa observada daí em diante como saída. Isso significa que reduzimos a aprendizagem por reforço a um problema de aprendizagem indutiva padrão, conforme discutimos no Capítulo 18. A Seção 21.4 descreve a utilização de tipos mais poderosos de representações para a função utilidade, como redes neurais. As técnicas de aprendizagem para essas representações podem ser aplicadas diretamente aos dados observados.

A estimativa de utilidade direta tem sucesso na redução do problema de aprendizagem por reforço a um problema de aprendizagem indutiva sobre o que já se sabe muito. Infelizmente, ela omite uma fonte de informação muito importante, ou seja, o fato de que as utilidades de estados não são independentes! *A utilidade de cada estado é igual à sua própria recompensa, somada à utilidade esperada de seus estados sucessores*. Isto é, os valores de utilidade obedecem às equações de Bellman para uma política fixa (veja também a Equação 17.10):

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^{\pi}(s') \,. \tag{21.2}$$

Ignorando as conexões entre estados, a estimativa de utilidade direta perde oportunidades para a

aprendizagem. Por exemplo, a segunda das três sequências de experiências (trials) dadas anteriormente alcança o estado (3,2), que não foi visitado antes. A próxima transição alcança (3,3), conhecido na primeira sequência (trial) como um estado que tem alta utilidade. A equação de Bellman sugere imediatamente que também é provável que (3,2) tenha alta utilidade porque leva a (3,3), mas a estimativa de utilidade direta não aprende nada até o fim do teste. Em termos mais amplos, podemos visualizar a estimativa de utilidade direta como a busca em um espaço de hipóteses para *U* muito maior do que precisa ser, no sentido de incluir muitas funções que violam as equações de Bellman. Por essa razão, o algoritmo frequentemente converge de forma muito lenta.

21.2.2 Programação dinâmica adaptativa

Um agente de **programação dinâmica adaptativa** (ou **PDA**) leva vantagem das restrições entre as utilidades de estados aprendendo o modelo de transição que os conecta e resolvendo o processo de decisão de Markov correspondente, utilizando um método de programação dinâmica. Para um agente de aprendizagem passiva, isso significa inserir o modelo de transição aprendido $P(s' \mid s, \text{p}(s))$ e as recompensas observadas $R(s)$ nas equações de Bellman (21.2) para calcular as utilidades dos estados. Conforme comentamos em nossa descrição do algoritmo de iteração de política no Capítulo 17, essas equações são lineares (não há nenhuma maximização envolvida), e, assim, elas podem ser resolvidas com a utilização de qualquer pacote de álgebra linear. Como alternativa, podemos adotar a abordagem de **iteração de política modificada** (Seção 17.3), usando um processo de iteração de valor simplificado para atualizar as estimativas de utilidade depois de cada mudança no modelo aprendido. Tendo em vista que, em geral, o modelo só muda ligeiramente a cada observação, o processo de iteração de valor pode empregar as estimativas de utilidade anteriores como valores iniciais e deve convergir com bastante rapidez.

O processo de aprender o modelo em si é fácil porque o ambiente é completamente observável. Isso significa que temos uma tarefa de aprendizagem supervisionada em que a entrada é um par estado-ação e a saída é o estado resultante. No caso mais simples, podemos representar o modelo de transição como uma tabela de probabilidades. Mantemos o controle da frequência com que ocorre cada resultado de ação e avaliamos a probabilidade de transição $P(s', s, a)$ a partir da frequência com que s' é alcançado quando se executa a em s. Por exemplo, nos três testes representados, *Direita* é executada três vezes em (1,3) e em duas das três vezes o estado resultante é (2,3); assim, $P((2,3) \mid (1,3), \textit{Direita})$ é avaliado como 2/3.

O programa de agente completo para um agente de PDA passivo é mostrado na Figura 21.2. Seu desempenho no mundo 4 × 3 é apresentado na Figura 21.3. Em termos da rapidez com que suas estimativas de valores melhoram, o agente de PDA é limitado apenas por sua habilidade de aprender o modelo de transição. Nesse sentido, ele fornece um padrão de comparação para outros algoritmos de aprendizagem por reforço. Entretanto, ele é um tanto intratável para grandes espaços de estados. Por exemplo, em gamão, ele envolveria a resolução de aproximadamente 10^{50} equações com 10^{50} incógnitas.

função AGENTE-PDA-PASSIVO(*percepção*) **retorna** uma ação

entradas: *percepção*, uma percepção indicando o estado atual s' e o sinal de recompensa r'
variáveis estáticas: π, uma política fixa
mdp, um MDP com modelo T, recompensas R e desconto γ
U, uma tabela de utilidades, inicialmente vazia
N_{sa}, uma tabela de frequências referente aos pares estado-ação, inicialmente zero
$Ns_{'|}as$, uma tabela de frequências correspondentes a triplas estado-ação-estado, inicialmente zero
s, a, estado e ação anteriores, inicialmente nulos
se s' é novo **então faça** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
se s é não nulo **então faça**
incrementar $N_{sa}[s, a]$ e $Ns_{'|}as\ [s', s, a]$
para cada t tal que $N_{s'|s,a}\ [t, s, a]$ é diferente de zero **faça**
$P(t, s, a) \leftarrow Ns_{'|}as\ [t, s, a] \ / \ N_{sa}\ [s, \text{a}]$
$U \leftarrow$ AVALIAÇÃO-DE-POLÍTICA(p, U, *mdp*)
se s'.TERMINAL? **então** $s, a \leftarrow$ nulo **senão** $s, a \leftarrow s', \pi[s']$
retornar a

Figura 21.2 Agente de aprendizagem por reforço passivo baseado em programação dinâmica adaptativa. A função de AVALIAÇÃO-DE -POLÍTICA resolve as equações de Bellman de política fixa, como descrito na Seção 17.3.

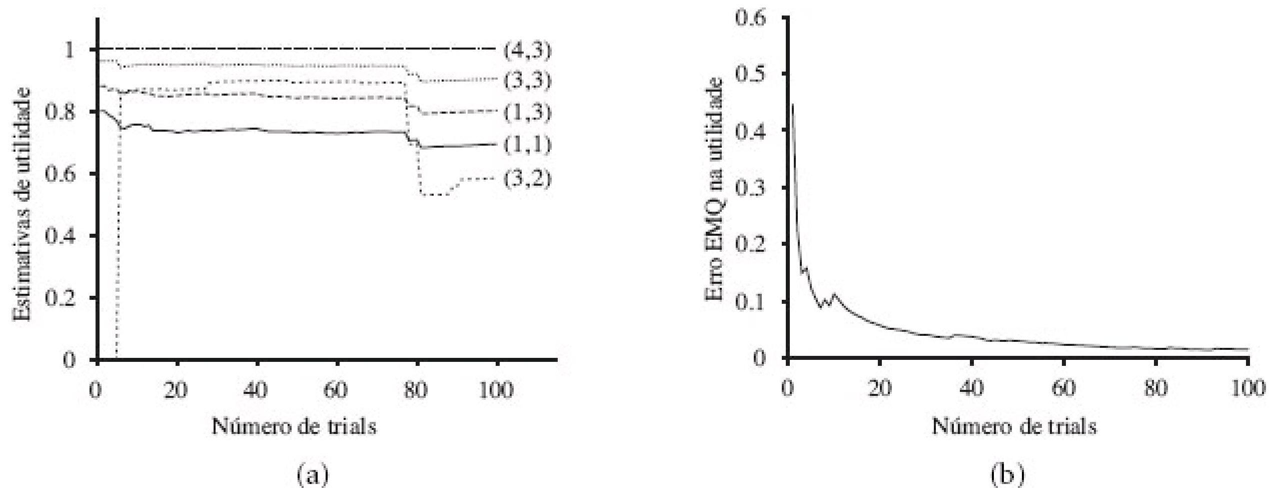


Figura 21.3 Curvas de aprendizagem de PDA passivo para o mundo 4 × 3, dada a política ótima mostrada na Figura 21.1. (a) Estimativas de utilidade para um subconjunto selecionado de estados, como uma função do número de experiências. Note as grandes mudanças que ocorrem em torno da 78º experiência (trial) – essa é a primeira vez que o agente cai no estado terminal –1 em (4,2). (b) Erro médio quadrático (EMQ) (Apêndice A) na estimativa de $U(1,1)$, calculado sobre 20 execuções de 100 trials cada.

Um leitor familiarizado com as ideias de aprendizagem bayesianas do Capítulo 20 deve ter notado que o algoritmo na Figura 21.2 está usando uma estimativa de probabilidade máxima para aprender o

modelo de transição; além disso, ao escolher uma política baseada unicamente no modelo *estimado*, o agente está agindo *como se* o modelo estivesse correto. Isso não é necessariamente uma boa ideia! Por exemplo, um agente de táxi que não sabe sobre como os semáforos podem ignorar uma luz vermelha uma ou duas vezes, sem nenhum efeito catastrófico e, em seguida, e a partir de então, formular uma política para ignorar as luzes vermelhas. Em vez disso, pode ser uma boa ideia escolher uma política que, embora não seja ótima para o modelo estimado pela probabilidade máxima, funcione razoavelmente bem para toda a gama de modelos que têm uma chance razoável de ser o verdadeiro modelo. Existem duas abordagens matemáticas que usam essa ideia.

A primeira abordagem, o **aprendizado por reforço bayesiano**, assume uma probabilidade *a priori* $P(h)$ para cada hipótese h sobre o que é o verdadeiro modelo; a probabilidade posterior $P(h \mid \mathbf{e})$ é obtida da maneira usual pela regra de Bayes dadas as observações até o momento. Então, se o agente decidiu parar de aprender, a política ótima é aquela que fornece a maior utilidade esperada. Seja u_h^π a utilidade esperada, rateada sobre todos os estados iniciais possíveis, obtidos pela execução da política no modelo h. Então temos

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_h P(h \mid \mathbf{e}) u_h^\pi .$$

Em alguns casos especiais, essa política pode até ser calculada! No entanto, se o agente continuar a aprender no futuro, encontrar uma política ótima torna-se consideravelmente mais difícil porque o agente deve considerar os efeitos de futuras observações sobre suas crenças acerca do modelo de transição. O problema torna-se um POMDP cujos estados de crença são distribuições sobre os modelos. Esse conceito fornece uma base analítica para a compreensão do problema de exploração descrito na Seção 21.3.

A segunda abordagem, derivada da **teoria de controle robusto**, permite um conjunto de controles possíveis $\mathcal{H}$ e define uma política robusta ótima como aquela que fornece o melhor resultado no *pior caso* de $\mathcal{H}$:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \min_h u_h^\pi .$$

Muitas vezes, o conjunto $\mathcal{H}$ será o conjunto de modelos que ultrapassam algum limiar de probabilidade em $P(h \mid \mathbf{e})$; assim, as abordagens robusta e bayesiana estão relacionadas. Às vezes, a solução robusta pode ser calculada de forma eficiente. Além disso, há os algoritmos de reforço de aprendizagem que tendem a produzir soluções robustas, mas não os abrangeremos aqui.

21.2.3 Aprendizagem de diferença temporal

Resolver o MDP subjacente, como na seção anterior, não é a única maneira de usar as equações de Bellman para lidar com o problema de aprendizagem. Outra maneira é usar as transições observadas para ajustar as utilidades dos estados observados, de forma que eles concordem com as equações de restrições. Por exemplo, considere a transição de (1,3) para (2,3) no segundo trial do início desse capítulo. Suponha que, como resultado do primeiro teste, as estimativas de utilidade sejam $U^\pi(1,3) =$

0,84 e $U^{\pi}(2,3) = 0,92$. Agora, se essa transição ocorresse durante todo o tempo, esperaríamos que as utilidades obedecessem à equação

$$U^{\pi}(1,3) = -0,04 + U^{\pi}(2,3),$$

e, então, $U^{\pi}(1,3)$ seria 0,88. Desse modo, sua estimativa atual de 0,84 talvez esteja um pouco baixa e deva ser aumentada. De modo mais geral, quando ocorre uma transição do estado s para o estado s', aplicamos a seguinte atualização a $U^{\pi}(s)$:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma\, U^{\pi}(s') - U^{\pi}(s)). \qquad (21.3)$$

Aqui, α é o parâmetro de **taxa de aprendizagem**. Como essa regra de atualização emprega a diferença de utilidades entre estados sucessivos, com frequência ela é chamada de equação de **diferença temporal** ou DT.

Todos os métodos de diferença temporal trabalham ajustando as estimativas de utilidade para o equilíbrio ideal que é válido localmente, quando as estimativas de utilidade estão corretas. No caso da aprendizagem passiva, o equilíbrio é dado pela Equação 21.2. Agora, a Equação 21.3 faz de fato o agente alcançar o equilíbrio dado pela Equação 21.2, mas existe certa sutileza envolvida. Primeiro, note que a atualização envolve apenas o sucessor observado s', enquanto as condições reais de equilíbrio envolvem todos os estados seguintes possíveis. Poderíamos imaginar que isso causasse uma mudança inapropriadamente grande em $U^{\pi}(s)$ quando ocorresse uma transição muito rara; mas, de fato, como transições raras ocorrem apenas raramente, o *valor médio* de $U^{\pi}(s)$ convergirá para o valor correto. Além disso, se mudarmos α de um parâmetro fixo para uma função que decresce à medida que aumenta o número de vezes em que um estado é visitado, então a própria $U^{\pi}(s)$ convergirá para o valor correto.[1] Isso nos dá o programa de agente mostrado na Figura 21.4. A Figura 21.5 ilustra o desempenho do agente de DT passivo no mundo 4 × 3. Ele não aprende tão rápido quanto o agente de PDA e mostra variabilidade muito mais alta, mas é muito mais simples e exige muito menos computação por observação. Note que *DT não precisa de um modelo para executar suas atualizações*. O ambiente fornece a conexão entre estados vizinhos sob a forma de transições observadas.

função AGENTE-DT-PASSIVO(*percepção*) **retorna** uma ação
  entradas: *percepção*, uma percepção indicando o estado atual s' e o sinal de recompensa r'
  variáveis estáticas: π, uma política fixa
    U, uma tabela de utilidades, inicialmente vazia
    N_s, uma tabela de frequências para estados, inicialmente zero
    s, a, r, estado, ação e recompensa anteriores, inicialmente nulos
  se s' é novo **então** $U[s'] \leftarrow r'$
  se s é não nulo **então**
    incrementar $N_s[s]$
      $U[s] \leftarrow U[s] + a(N_s[s])\ (r + \gamma\ U[s'] - U[s])$
  se TERMINAL?[s'] **então** $s, a, r \leftarrow$ nulo **senão** $s, a, r \leftarrow s', p[s'], r'$

retornar *a*

Figura 21.4 Um agente de aprendizagem por reforço que aprende estimativas de utilidade com diferenças temporais. Escolhe-se a função tamanho do passo a(n) para assegurar a convergência, como descrito no texto.

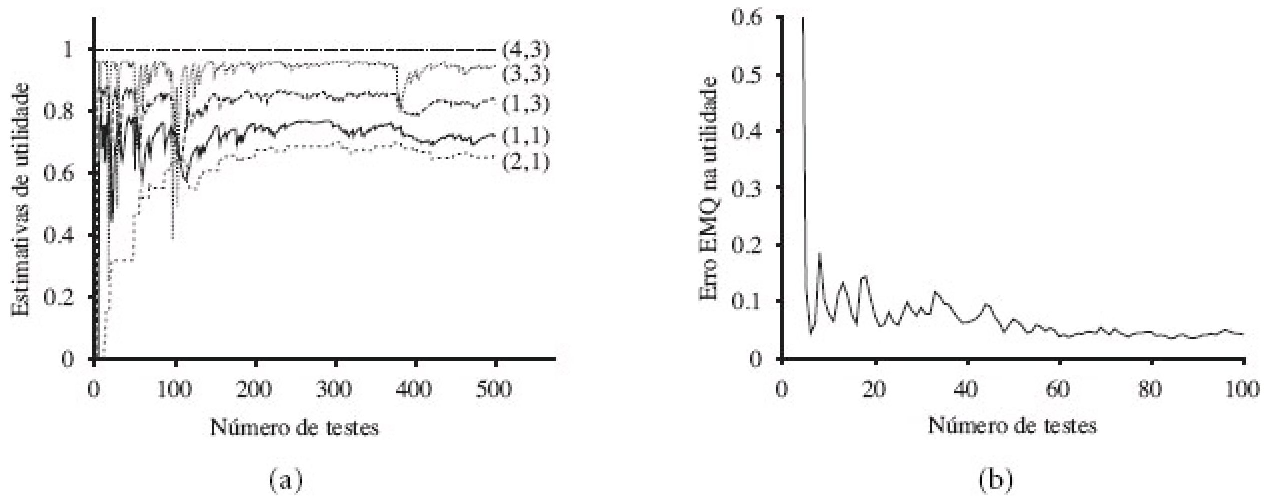


Figura 21.5 As curvas de aprendizagem de DTpara o mundo 4 × 3. (a) Estimativas de utilidade para um subconjunto selecionado de estados, como uma função do número de experiências (trials). (b) Erro médio quadrático na estimativa de *U*(1,1), calculado sobre 20 execuções de 500 trials cada. Apenas os 100 primeiros trials são mostrados, a fim de permitir a comparação com a Figura 21.3.

A abordagem de PDA e a abordagem de DT estão na realidade estreitamente relacionadas. Ambas tentam fazer ajustes locais para as estimativas de utilidade, a fim de fazer cada estado "concordar" com seus sucessores. Uma diferença é que DT ajusta um estado para concordar com seu sucessor *observado* (Equação 21.3), enquanto PDA ajusta o estado para concordar com *todos* os sucessores que poderiam ocorrer, ponderados por suas probabilidades (Equação 21.2). Essa diferença desaparece quando os efeitos dos ajustes de DT têm sua média calculada sobre um grande número de transições porque a frequência de cada sucessor no conjunto de transições é aproximadamente proporcional à sua probabilidade. Uma diferença mais importante é que, enquanto DT faz um único ajuste por transição observada, PDA faz tantos quantos necessita para restaurar a consistência entre as estimativas de utilidade *U* e o modelo de ambiente *P*. Embora a transição observada faça apenas uma mudança local em *P*, seus efeitos talvez tenham de ser propagados ao longo de *U*. Desse modo, DT pode ser visualizada como uma primeira aproximação, crua mas eficiente, para PDA.

Cada ajuste feito por PDA pode ser considerado, do ponto de vista de DT, como resultado de uma "pseudoexperiência" gerada pela simulação do modelo de ambiente atual. É possível estender a abordagem de DT para usar um modelo de ambiente que gere várias pseudoexperiências — transições que o agente de DT talvez imagine que *poderiam* acontecer, dado seu modelo atual. Para cada transição observada, o agente de DT pode gerar grande número de transições imaginárias. Desse modo, as estimativas de utilidade resultantes se aproximarão cada vez mais das estimativas de PDA — é claro, a um preço de um tempo maior de computação.

De modo semelhante, podemos gerar versões mais eficientes de PDA pela aproximação direta dos algoritmos de iteração de valor ou iteração de política. Mesmo que o algoritmo de iteração de valor

seja eficiente, ele é intratável se tivermos, digamos, 10^{100} estados. No entanto, muitos dos ajustes necessários para os valores de estado em cada iteração serão extremamente pequenos. Uma abordagem possível para gerar respostas de qualidade razoável com rapidez é limitar o número de ajustes feitos depois de cada transição observada. Também poderíamos utilizar uma heurística para ordenar os ajustes possíveis de modo a executar apenas os mais significativos. A heurística de **varredura priorizada** prefere fazer ajustes em estados cujos *prováveis* sucessores acabaram de sofrer um *grande* ajuste em suas próprias estimativas de utilidade. Usando heurísticas como essa, os algoritmos de PDA aproximada em geral podem aprender quase tão rápido quanto a PDA completa, em termos do número de sequências de treinamento, mas podem ser várias ordens de magnitude mais eficientes em termos de computação (veja o Exercício 21.3). Isso lhes permite manipular espaços de estados que são muito maiores para a PDA completa. Os algoritmos de PDA aproximada têm uma vantagem adicional: nas fases iniciais de aprendizagem de um novo ambiente, o modelo de ambiente P com frequência estará longe de ser correto e, assim, haverá pouca razão para calcular uma função utilidade exata que corresponda a esse modelo. Um algoritmo de aproximação pode usar um tamanho de ajuste mínimo que diminui à medida que o modelo do ambiente se torna mais preciso. Isso elimina as iterações de valor muito longas que podem ocorrer nas fases iniciais de aprendizagem em consequência de grandes mudanças no modelo.

21.3 APRENDIZAGEM POR REFORÇO ATIVA

Um agente de aprendizagem passiva tem uma política fixa que determina seu comportamento. Um agente de aprendizagem ativa deve decidir que ações executar. Vamos começar com o agente de programação dinâmica adaptativa e considerar o modo como ele deve ser modificado para tratar essa nova liberdade.

Primeiro, o agente precisará aprender um modelo completo com probabilidades de resultados para todas as ações, em vez de aprender apenas o modelo para a política fixa. O mecanismo de aprendizagem simples usado por AGENTE-PDA-PASSIVO funcionará muito bem para isso. Em seguida, precisaremos levar em conta o fato de que o agente tem uma escolha de ações. As utilidades que ele precisa aprender são aquelas definidas pela política *ótima*; elas obedecem às equações de Bellman dadas anteriormente (Seção 17.2.1), que repetiremos aqui por conveniência:

$$U(s) = R(s) + \gamma \max_{a} \sum_{s'} P(s' \mid s, a) U(s') . \tag{21.4}$$

Essas equações podem ser resolvidas para se obter a função utilidade U empregando os algoritmos de iteração de valor ou iteração de política do Capítulo 17. A última questão é o que fazer em cada etapa. Tendo obtido uma função utilidade U ótima para o modelo aprendido, o agente pode extrair uma ação ótima por meio da observação antecipada de um passo a fim de maximizar a utilidade esperada; como alternativa, se ele utilizar a iteração de política, a política ótima já estará disponível e, portanto, ele deverá simplesmente executar a ação que a política ótima recomendar. Ou não?

21.3.1 Exploração

A Figura 21.6 mostra os resultados de uma sequência de trials para um agente de PDA que segue a recomendação da política ótima para o modelo aprendido em cada etapa. O agente *não* aprende as utilidades verdadeiras ou a política ótima verdadeira! Em vez disso, o que acontece é que, no 39º trial, ele encontra uma política que alcança a recompensa +1 ao longo da rota inferior, via (2,1), (3,1), (3,2) e (3,3) (veja a Figura 21.6(b)). Depois de fazer experiências com pequenas variações, a partir do 276º trial em diante, ele se fixa nessa política, nunca aprendendo as utilidades dos outros estados e nunca encontrando a rota ótima via (1,2), (1,3) e (2,3). Damos a esse agente o nome de **agente guloso**. Experimentos repetidos mostram que o agente guloso *muito raramente* converge para a política ótima correspondente a esse ambiente e, às vezes, converge para políticas realmente ruins.

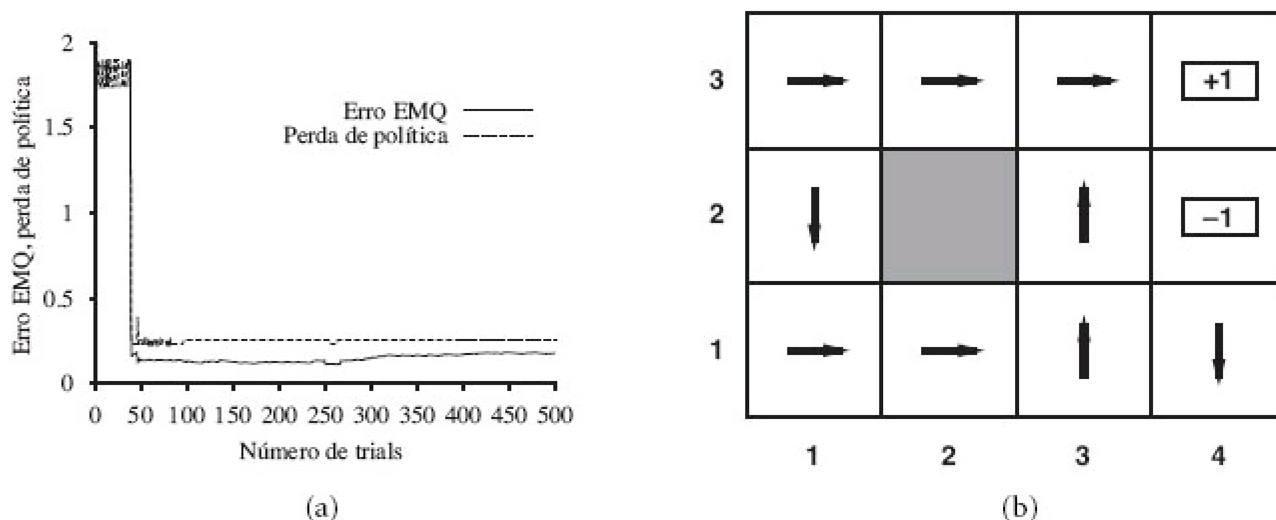


Figura 21.6 Desempenho de um agente de PDA guloso que executa a ação recomendada pela política ótima para o modelo aprendido. (a) Erro EMQ nas estimativas médias de utilidade calculadas sobre os nove quadrados não terminais. (b) Política não ótima para a qual o agente guloso converge nessa sequência específica de trials.

Como pode ocorrer de a escolha da ação ótima levar a resultados não ótimos? A resposta é que o modelo aprendido não é o mesmo do ambiente verdadeiro; o que é ótimo no modelo aprendido pode então ser não ótimo no ambiente verdadeiro. Infelizmente, o agente não sabe qual é o ambiente verdadeiro e, assim, ele não pode calcular a ação ótima para o ambiente verdadeiro. Então, o que deve ser feito?

O que o agente guloso deixou de considerar é que as ações fazem mais que fornecer recompensas de acordo com o modelo atual aprendido; elas também contribuem para a aprendizagem do modelo verdadeiro, afetando as percepções que são recebidas. Melhorando o modelo, o agente receberá recompensas maiores no futuro.[2] Portanto, um agente deve assumir um compromisso entre **exploitação** (aproveitamento) para maximizar sua recompensa — que se reflete em suas estimativas de utilidade atuais — e **exploração**, a fim de maximizar seu bem-estar a longo prazo. A exploitação puro se arrisca a ficar paralisado em um beco. A exploração pura para melhorar o conhecimento é inútil se não colocamos em prática esse conhecimento. No mundo real, constantemente é preciso decidir entre continuar a manter uma existência confortável e mergulhar no desconhecido, na esperança de descobrir uma vida nova e melhor. Com maior compreensão, é necessário menos

exploração.

Podemos ser um pouco mais precisos do que isso? Existe uma política *ótima* de exploração? Ocorre que essa questão foi estudada em profundidade no subcampo da teoria da decisão estatística que lida com os chamados **problemas do bandido**.

Embora os problemas do bandido sejam extremamente difíceis de resolver com exatidão para se obter um método de exploração *ótimo*, é possível apresentar um esquema *razoável* que vai eventualmente levar a um comportamento ótimo do agente. Tecnicamente, qualquer esquema desse tipo precisará ser guloso no limite da exploração infinita, ou **GLIE** (*greedy in the limit of infinite exploration*). Um esquema GLIE deve experimentar cada ação em cada estado um número ilimitado de vezes, para evitar ter probabilidade finita de que uma ação ótima seja omitida devido a uma série de resultados de qualidade excepcionalmente ruim. Um agente de PDA usando tal esquema eventualmente vai aprender o modelo do ambiente verdadeiro. Um esquema GLIE também deve eventualmente se tornar guloso, de modo que as ações do agente se tornem ótimas em relação ao modelo aprendido (e, portanto, verdadeiro).

Existem vários esquemas de GLIE; um dos mais simples é fazer o agente escolher uma ação ao acaso durante uma fração $1/t$ do tempo e seguir a política gulosa em caso contrário. Embora isso eventualmente possa convergir para uma política ótima, pode ser muito lento. Uma abordagem mais sensata daria algum peso a ações que o agente não tentou com muita frequência, enquanto tenderia a evitar ações consideradas de baixa utilidade. Isso pode ser implementado alterando-se a equação de restrição (21.4), de forma que ela atribua uma estimativa de utilidade mais alta a pares estado-ação relativamente inexplorados. Em essência, isso leva a um *a priori* otimista sobre os ambientes possíveis e faz o agente se comportar no início como se houvesse recompensas maravilhosas espalhadas por todos os lados. Vamos usar $U^+(s)$ para denotar a estimativa otimista da utilidade (isto é, a recompensa esperada desse momento em diante) do estado s, e seja $N(s, a)$ o número de vezes que a ação a foi tentada no estado s. Vamos supor que estamos usando a iteração de valor em um agente de aprendizagem de PDA; então, precisamos reescrever a equação de atualização (isto é, a Equação 17.6 para incorporar a estimativa otimista. A equação a seguir faz isso:

$$U^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} P(s' \mid s, a) U^+(s'),\ N(s, a)\right) . \quad (21.5)$$

EXPLORAÇÃO E BANDIDOS

Em Las Vegas, *um bandido com um braço* é uma máquina caça-níqueis. Um jogador pode inserir uma moeda, puxar a alavanca e recolher os ganhos (se houver). Um **bandido** com ***n*-braços** tem n alavancas. O jogador deve escolher qual alavanca acionar em cada moeda sucessiva — aquela que pagou a melhor recompensa ou, quem sabe, a que ainda não foi experimentada.

O problema do bandido com n-braços é um modelo formal para problemas reais em muitas áreas de importância vital, como a decisão sobre o orçamento anual para pesquisa e desenvolvimento em IA. Cada braço corresponde a uma ação (como a de alocar $20 milhões para o desenvolvimento de novos livros didáticos sobre IA) e a recompensa de puxar a alavanca correspondente aos benefícios (imensos) obtidos a partir da execução da ação. A exploração, quer

seja a exploração de um novo campo de pesquisa ou a de um novo centro comercial, é arriscada, dispendiosa e tem recompensas incertas; por outro lado, deixar de explorar significa que nunca se descobrirá *alguma* ação que valha a pena.

Para formular um problema de bandido corretamente, devemos definir com exatidão o que queremos dizer com comportamento ótimo. A maior parte das definições na literatura supõe que o objetivo é maximizar a recompensa total esperada obtida ao longo do tempo de duração do agente. Essas definições exigem que a expectativa seja considerada sobre os mundos possíveis em que o agente poderia estar, bem como sobre os resultados possíveis de cada sequência de ações em qualquer mundo dado. Aqui, um "mundo" é definido pelo modelo de transição $P(s', s, a)$. Desse modo, para agir de forma ótima, o agente precisa de uma distribuição *a priori* sobre os modelos possíveis. Os problemas de otimização resultantes em geral são terrivelmente intratáveis.

Em alguns casos — por exemplo, quando o retorno de cada máquina é independente e são usadas recompensas descontadas —, é possível calcular um **índice Gittins** para cada máquina caça-níqueis (Gittins, 1989). O índice é uma função apenas do número de vezes que a máquina caça-níqueis foi usada em jogos e de quanto ela retornou de recompensa. O índice para cada máquina indica o quanto vale a pena investir mais; falando de modo geral, quanto maior o retorno esperado e a incerteza na utilidade de uma escolha, melhor. A escolha da máquina com o valor de índice mais alto fornece uma política de exploração ótima. Infelizmente, não foi encontrado nenhum modo de estender os índices de Gittins a problemas de decisão sequencial.

Podemos usar a teoria de bandidos com *n*-braços para discutir a racionalidade da estratégia de seleção em algoritmos genéticos (veja o Capítulo 4). Se você considerar cada braço em um problema de bandido com *n*-braços uma sequência possível de genes, pode ser provado que os algoritmos genéticos vão alocar moedas de forma ótima, dado um conjunto apropriado de suposições de independência.

Aqui, $f(u, n)$ é chamada **função de exploração**. Ela determina como a gula (preferência por altos valores de u) é equilibrada pela curiosidade (preferência por ações que não foram experimentadas com frequência e têm valores baixos de n). A função $f(u, n)$ deve ser crescente em u e decrescente em n. É óbvio que existem muitas funções possíveis que se ajustam a essas condições. Uma definição particularmente simples é:

$$f(u,n) = \begin{cases} R^+ & \text{se } n < N_e \\ u & \text{em caso contrário} \end{cases}$$

onde $R+$ é uma estimativa otimista da melhor recompensa possível que pode ser obtida em qualquer estado, e N_e é um parâmetro fixo. Isso terá o efeito de fazer o agente experimentar cada par ação-estado pelo menos N_e vezes.

O fato de U^+ aparecer em lugar de U no lado direito da Equação 21.5 é muito importante. À medida que a exploração prossegue, os estados e as ações próximos ao estado inicial podem ser experimentados um grande número de vezes. Se usássemos U, a estimativa de utilidade mais pessimista, o agente logo se tornaria pouco inclinado a explorar ainda mais profundamente no campo. O uso de U^+ quer dizer que os benefícios de exploração são propagados a partir das fronteiras de

regiões inexploradas, de forma que as ações que levem *em direção a* regiões inexploradas sejam ponderadas com peso mais alto, e não apenas ações que são pouco familiares. O efeito dessa política de exploração pode ser visto claramente na Figura 21.7, que mostra uma rápida convergência em direção ao desempenho ótimo, diferentemente do que acontece na abordagem gulosa. Uma política muito próxima da ótima é encontrada depois de apenas 18 trials. Note que as próprias estimativas de utilidade não convergem com tanta rapidez. Isso ocorre porque o agente logo interrompe a exploração das partes pouco compensadoras do espaço de estados, visitando-as somente "por acaso" daí em diante. Porém, faz sentido para o agente não se importar com as utilidades exatas de estados que ele sabe que são indesejáveis e que podem ser evitadas.

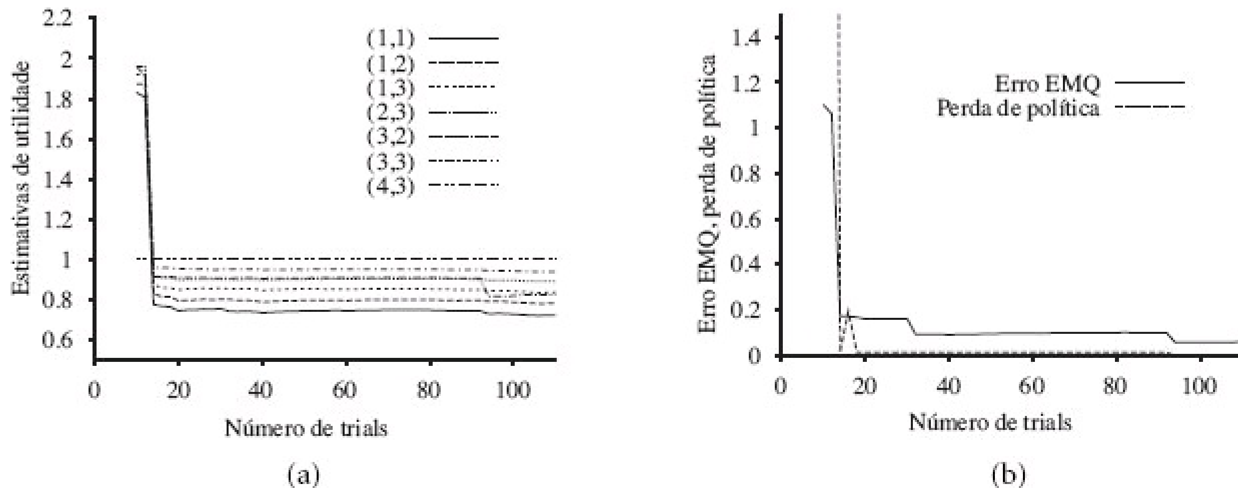


Figura 21.7 Desempenho do agente de PDA de exploração, utilizando-se $R^+ = 2$ e $N_e = 5$. (a) Estimativas de utilidade para estados selecionados ao longo do tempo. (b) O erro EMQ em valores de utilidade e a perda de política associada.

21.3.2 Aprendizagem de uma função de ação-valor

Agora, que temos um agente de PDA ativo, vamos considerar como construir um agente de aprendizagem ativo de diferença temporal. A mudança mais óbvia em relação ao caso passivo é que o agente não está mais equipado com uma política fixa e, assim, se aprender uma função utilidade U, ele precisará aprender um modelo para ser capaz de escolher uma ação baseada em U por meio da observação antecipada de um passo. O problema de aquisição de modelo para o agente de DT é idêntico ao do agente de PDA. E a regra de atualização do DT em si? Talvez pareça surpreendente o fato de a regra de atualização (21.3) permanecer inalterada. É possível que isso pareça estranho, pela seguinte razão: suponha que o agente execute um passo que normalmente leva a um bom destino, mas, devido ao não determinismo no ambiente, o agente acaba em um estado catastrófico. A regra de atualização de DT levará essa situação tão a sério como se seu resultado fosse o resultado normal da ação, embora se possa supor que, devido ao fato de o resultado ter sido obtido por casualidade, o agente não deve se preocupar muito com ele. De fato, é claro que o resultado improvável ocorrerá com pouca frequência em um grande conjunto de sequências de treinamento; por conseguinte, em longo prazo, seus efeitos serão ponderados de forma proporcional à sua probabilidade, conforme seria de esperar. Mais uma vez, pode-se mostrar que o algoritmo de DT convergirá para os mesmos valores que a PDA, à medida que o número de sequências de treinamento tender a infinito.

Existe um método de DT alternativo chamado **aprendizagem *Q*** que aprende uma representação de ação-utilidade, em vez de aprender utilidades. Usaremos a notação $Q(s, a)$ para denotar o valor da execução da ação *a* no estado *s*. Os valores de Q estão diretamente relacionados a valores de utilidade, como a seguir:

$$U(s) = \max_a Q(s, a) . \qquad (21.6)$$

As funções *Q* podem parecer apenas outro modo de armazenar informações de utilidade, mas elas têm uma propriedade muito importante: *um agente de DT que aprende uma função Q não precisa de um modelo da forma P(s' | s,a), para aprendizagem ou seleção de ações*. Por essa razão, a aprendizagem *Q* é chamada de método **livre de modelo**. Como ocorre no caso das utilidades, podemos escrever uma equação de restrição que deve se manter em equilíbrio quando os valores de *Q* estiverem corretos:

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q(s', a') . \qquad (21.7)$$

Como no agente de aprendizagem de PDA, podemos usar essa equação diretamente como uma equação de atualização para um processo de iteração que calcula valores de *Q* exatos, dado um modelo estimado. Porém, isso exige que um modelo também seja aprendido, porque a equação utiliza $P(s' \square s, a)$. Por outro lado, a abordagem de diferença temporal não exige nenhum modelo de transições de estado — tudo o que é preciso são os valores de *Q*. A equação de atualização para a aprendizagemQ de DT é:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)) , \qquad (21.8)$$

calculada sempre que a ação *a* é executada no estado *s* e leva ao estado *s'*.

O projeto de agente completo para um agente exploratório de aprendizagem *Q* usando DT é mostrado na Figura 21.8. Note que ele utiliza exatamente a mesma função de exploração *f* que foi utilizada pelo agente exploratório de PDA — daí a necessidade de manter estatísticas sobre ações executadas (a tabela *N*). Se for usada uma política de exploração mais simples — digamos, agir ao acaso sobre alguma fração de passos, onde a fração decresce com o passar do tempo — poderemos dispensar a estatística.

função AGENTE-DE-APRENDIZAGEM-Q(*percepção*) **retorna** uma ação
  entradas: *percepção*, uma percepção que indica o estado atual *s'* e o sinal de recompensa *r'*
  variáveis estáticas: *Q*, uma tabela de valores de ações indexada por estado e ação, inicializada com zero
    N_{sa}, uma tabela de frequências correspondentes a pares estado-ação, inicializada com zero
    s, *a*, *r*, estado, ação e recompensa anteriores, inicialmente nulos
  se *s*.TERMINAL? **então** *Q*[*s, Nome*] ← *r'*
  se *s* é não nulo **então**

incrementar $N_{sa}[s, a]$
$Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] \quad Q[s, a])$
$s, a, r \leftarrow s'$, $\text{argmax}_{a'}, f(Q[s', a'], N_{sa}[s', a']), r'$
retornar *a*

Figura 21.8 Um agente exploratório de aprendizagem *Q*. Esse é um agente de aprendizagem ativo que aprende o valor $Q(s, a)$ de cada ação em cada situação. Ele utiliza a mesma função de exploração *f* que o agente exploratório de PDA, mas evita ter de aprender o modelo de transição porque o valor de *Q* para um estado pode ser relacionado diretamente aos de seus vizinhos.

A aprendizagem *Q* tem um parente próximo chamado **SARSA** (State-Action-Reward-Action). A regra de atualização para SARSA é muito semelhante à Equação 21.8:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma\, Q(s',a') - Q(s,a))\,, \tag{21.9}$$

onde a' é a ação *realmente tomada* no estado s'. A regra é aplicada no final de cada quíntuplo s, a, r, s', a', daí o nome. A diferença da aprendizagem *Q* é bastante sutil: enquanto a aprendizagem *Q* copia o *melhor* valor de *Q* do estado atingido na transição observada, SARSA espera até que uma ação seja tomada realmente e copia o valor de *Q* dessa ação. Agora, para um agente guloso que sempre toma a ação com o melhor valor de *Q*, os dois algoritmos são idênticos. Quando a exploração estiver acontecendo, no entanto, eles diferem significativamente. Como a aprendizagem *Q* utiliza o melhor valor de *Q*, não presta atenção à política real que está sendo seguida — é um algoritmo de aprendizagem **fora da política**, enquanto SARSA é um algoritmo **dentro da política**. A aprendizagem *Q* é mais flexível do que SARSA, no sentido de que um agente de aprendizagem *Q* pode aprender como se comportar bem, mesmo quando orientado por uma política de exploração aleatória ou adversarial. Por outro lado, SARSA é mais realista: por exemplo, se a política global estiver ainda parcialmente controlada por outros agentes, é melhor aprender uma função *Q* para o que vai acontecer realmente e não para o que o agente gostaria que acontecesse.

Tanto a aprendizagem *Q* como SARSA aprendem a política ótima para o mundo 4×3, mas o fazem a uma taxa muito mais lenta que o agente de PDA. Isso ocorre porque a atualização local não impõe a consistência entre todos os valores de *Q* por meio do modelo. A comparação desperta uma questão geral: é melhor aprender um modelo e uma função utilidade ou aprender uma função de ação-valor sem modelo? Em outras palavras, qual é a melhor forma de representar a função do agente? Essa é uma questão encontrada nos fundamentos da inteligência artificial. Conforme declaramos no Capítulo 1, uma das características históricas-chave de grande parte da pesquisa de IA é sua preferência por uma (muitas vezes não declarada) abordagem baseada em **conhecimento**. Isso nos leva à suposição de que o melhor modo de representar a função de agente é construir uma representação de alguns aspectos do ambiente em que o agente está situado.

Alguns pesquisadores, tanto dentro quanto fora da IA, afirmam que a disponibilidade de métodos livres de modelos como a aprendizagem *Q* significa que a abordagem baseada em conhecimento é desnecessária. Entretanto, não existem argumentos mais fortes além da intuição. Nossa intuição é que, à medida que o ambiente se torna mais complexo, as vantagens de uma abordagem de conhecimento ficam mais aparentes. Isso é válido até mesmo em jogos como xadrez, damas e gamão (veja a

próxima seção), nos quais o esforço para aprender uma função de avaliação por meio de um modelo tiveram muito mais sucesso que os métodos de aprendizagem Q.

21.4 GENERALIZAÇÃO DA APRENDIZAGEM POR REFORÇO

Até agora, presumimos que as funções utilidade e as funções Q aprendidas pelos agentes são representadas em forma tabular com um valor de saída para cada tupla de entrada. Tal abordagem funciona razoavelmente bem para pequenos espaços de estados, mas o tempo para convergência e o tempo por iteração (para PDA) aumentam com rapidez à medida que o espaço fica maior. Com métodos de PDA aproximados e cuidadosamente controlados, talvez fosse possível tratar 10.000 estados ou mais. Isso basta para ambientes bidimensionais como labirintos, mas mundos mais realistas estão fora de questão. O xadrez e o gamão são minúsculos subconjuntos do mundo real, ainda que seus espaços de estados contenham um número de estados da ordem de 10^{20} e 10^{40} estados, repectivamente. Seria absurdo supor que alguém tivesse de visitar todos esses estados a fim de aprender a jogar!

Um modo de manipular tais problemas é usar a **aproximação de função**, que simplesmente significa usar qualquer tipo de representação para a função Q que não seja uma tabela de consulta. A representação é visualizada como aproximada porque talvez a função utilidade *verdadeira* ou a função Q não pudessem ser representadas na forma escolhida. Por exemplo, descrevemos no Capítulo 5 uma **função de avaliação** para xadrez que é representada sob a forma de função linear ponderada de um conjunto de **características** (ou de **funções base**) $f_1,\ldots,f_n$:

$$\hat{U}_\theta(s) = \theta_1\, f_1(s) + \theta_2\, f_2(s) + \cdots + \theta_n\, f_n(s)\,.$$

Uma aprendizagem por reforço pode aprender valores para os parâmetros $\theta = \theta_1,\ldots,\theta_n$ tais que a função de avaliação $\hat{U}\theta$ se aproxime da função utilidade verdadeira. Em vez de, digamos, 10^{40} valores em uma tabela, esse aproximador de função se caracteriza por, digamos, n = 20 parâmetros — uma *enorme* compactação. Embora ninguém conheça a função utilidade verdadeira para xadrez, ninguém acredita que ela possa ser representada exatamente em 20 números. No entanto, se a aproximação for boa o suficiente, talvez o agente ainda possa jogar xadrez muito bem.[3]

A aproximação de função torna prático representar funções utilidade para espaços de estados muito grandes, mas esse não é seu principal benefício. *A compactação alcançada por um aproximador de função permite ao agente de aprendizagem generalizar a partir de estados que ele visitou até estados que não visitou*. Ou seja, o aspecto mais importante da aproximação de função não é o fato de ela exigir menos espaço, mas o fato de permitir a generalização indutiva sobre estados de entrada. Para dar alguma ideia da capacidade desse efeito, observamos que, pelo exame de apenas um entre todos os 10^{12} estados possíveis de gamão, é possível aprender uma função utilidade que permita a um programa jogar tão bem quanto qualquer ser humano (Tesauro, 1992).

É claro que, por outro lado, existe um problema: talvez não seja possível haver qualquer função no espaço de hipóteses escolhido que se aproxime suficientemente bem da função utilidade verdadeira. Como em toda aprendizagem indutiva, existe um compromisso entre o tamanho do espaço de

hipóteses e o tempo necessário para aprender a função. Um espaço de hipóteses maior aumenta a probabilidade de ser encontrada uma boa aproximação, mas também significa que a convergência provavelmente será retardada.

Vamos começar com o caso mais simples, que é a estimativa de utilidade direta (veja a Seção 21.2). Com a aproximação de função, essa é uma instância de **aprendizagem supervisionada**. Por exemplo, suponha que representamos as utilidades para o mundo 4 × 3 usando uma função linear simples. As características dos quadrados são apenas suas coordenadas x e y, e assim temos:

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y \,. \tag{21.10}$$

Desse modo, se $(\theta_0, \theta_1, \theta_2) = (0{,}5,\ 0{,}2,\ 0{,}1)$, então $\hat{U}_\theta\,(1, 1) = 0{,}8$. Dada uma coleção de testes, obtemos um conjunto de valores de amostras de $\hat{U}_\theta(x, y)$ e podemos encontrar o melhor ajuste, no sentido de minimizar o erro quadrático, utilizando a regressão linear padrão (veja o Capítulo 18).

Para a aprendizagem por reforço, faz mais sentido usar um algoritmo de aprendizagem on-line que atualiza os parâmetros após cada trial. Vamos supor que executamos um trial e que a recompensa total obtida a partir de (1,1) seja 0,4. Isso sugere que $\hat{U}_\theta(1, 1)$, atualmente 0,8, é muito grande e deve ser reduzida. Como os parâmetros devem ser ajustados para se alcançar isso? Como ocorre na aprendizagem de redes neurais, escrevemos uma função de erro e calculamos seu gradiente em relação aos parâmetros. Se $u_j(s)$ é a recompensa total observada do estado s em diante no j-ésimo trial, então o erro é definido como (metade) da diferença quadrática entre o total previsto e o total real: $E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$. A taxa de mudança do erro em relação a cada parâmetro q_i é $\partial E_j/\partial\theta_i$; portanto, para mover o parâmetro na direção de diminuição do erro, queremos:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha\,(u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i} \,. \tag{21.11}$$

Isso se chama **regra deWidrow-Hoff**, ou **regra delta**, para mínimos quadrados on-line. No caso do aproximador de função linear $\hat{U}_\theta(s)$ na Equação 21.10, obtemos três regras de atualização simples:

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha\,(u_j(s) - \hat{U}_\theta(s)) \,, \\
\theta_1 &\leftarrow \theta_1 + \alpha\,(u_j(s) - \hat{U}_\theta(s))x \,, \\
\theta_2 &\leftarrow \theta_2 + \alpha\,(u_j(s) - \hat{U}_\theta(s))y \,.
\end{aligned}$$

Podemos aplicar essas regras ao exemplo em que $\hat{U}_\theta(1, 1)$ é 0,8 e $u_j(1,1)$ é 0,4. θ_0, θ_1 e θ_2 são reduzidos por 0,4α, o que reduz o erro para (1,1). Note que *a mudança dos parâmetros θ em resposta a uma transição observada entre dois estados também altera os valores de Ûθ para todos os outros estados*! Isso é o que queremos dizer ao afirmar que a aproximação de função permite a um agente de aprendizagem por reforço generalizar a partir de suas experiências.

Esperamos que o agente aprenda mais rápido se usar um aproximador de função, desde que o espaço de hipóteses não seja muito grande, mas inclua algumas funções que sejam um ajuste razoavelmente bom da função utilidade verdadeira. O Exercício 21.5 pede para avaliar o

desempenho da estimativa de utilidade direta, com e sem a função de aproximação. A melhoria no mundo 4 × 3 é notável, mas não drástica, porque esse é um espaço de estados muito pequeno para se iniciar. A melhoria é muito maior em um mundo 10 × 10 com recompensa a +1 em (10,10). Esse mundo é bem adequado para uma função utilidade linear porque a função utilidade verdadeira é suave e quase linear (veja o Exercício 21.8). Se colocarmos a recompensa +1 em (5,5), a utilidade verdadeira será semelhante a uma pirâmide e o aproximador de função na Equação 21.10 falhará totalmente. Porém, nem tudo está perdido! Lembramos que o que interessa no caso da aproximação de função linear é o fato de a função ser linear nos *parâmetros* — as próprias características podem ser funções não lineares arbitrárias das variáveis de estados. Consequentemente, podemos incluir uma expressão como $\theta_3 f_3(x, y) = \theta_3 \sqrt{(x - x_g)^2 + (y - y_g)^2}$, que mede a distância até a meta.

Podemos aplicar essas ideias igualmente bem a agentes de aprendizagem de diferença temporal. Tudo o que precisamos fazer é ajustar os parâmetros para tentar reduzir a diferença temporal entre estados sucessivos. As novas versões das equações de DT e de aprendizagem *Q* (21.3 e 21.8) são:

$$\theta_i \leftarrow \theta_i + \alpha\,[R(s) + \gamma\,\hat{U}_\theta(s') - \hat{U}_\theta(s)]\,\frac{\partial \hat{U}_\theta(s)}{\partial \theta_i} \tag{21.12}$$

para utilidades e

$$\theta_i \leftarrow \theta_i + \alpha\,[R(s) + \gamma\,\max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)]\,\frac{\partial \hat{Q}_\theta(s, a)}{\partial \theta_i} \tag{21.13}$$

para valores de *Q*. Para aprendizagem DTpassiva, a regra de atualização pode ser mostrada convergente para a maior aproximação possível[4] da função verdadeira quando o aproximador de função é *linear* nos parâmetros. Com aprendizagem ativa e funções *não lineares*, tais como redes neurais, todas as apostas estão fora. Existem alguns casos muito simples em que os parâmetros podem tender a infinito, embora haja boas soluções no espaço de hipóteses. Existem algoritmos mais sofisticados que podem evitar esses problemas, mas, no momento, a aprendizagem por reforço com aproximadores de funções gerais continua a ser uma arte delicada.

A aproximação de função também pode ser muito útil na aprendizagem de um modelo do ambiente. Lembre-se de que a aprendizagem de um modelo para um ambiente *observável* é um problema de aprendizagem *supervisionada* porque a próxima percepção dará o estado resultante. Quaisquer dos métodos de aprendizagem supervisionada do Capítulo 18 podem ser usados, com ajustes apropriados para o fato de que precisamos prever uma descrição de estados completa, em vez de simplesmente uma classificação booleana ou um único valor real. No caso de ambiente *parcialmente observável*, o problema de aprendizagem é muito mais difícil. Se soubermos quais são as variáveis ocultas e como elas mantêm um relacionamento causal umas com as outras e com as variáveis observáveis, poderemos fixar a estrutura de uma rede bayesiana dinâmica e utilizar o algoritmo EM para aprender os parâmetros, como descrevemos no Capítulo 20. A criação das variáveis ocultas e a aprendizagem da estrutura do modelo ainda são problemas em aberto.

Alguns exemplos práticos são descritos na Seção 21.6.

21.5 BUSCA DE POLÍTICAS

A última abordagem que examinaremos para problemas de aprendizagem por reforço é chamada **busca de políticas**. Em alguns aspectos, a busca de políticas é o mais simples de todos os métodos neste capítulo: a ideia é continuar ajustando a política enquanto isso fizer seu desempenho melhorar e então parar.

Vamos começar com as políticas propriamente ditas. Lembre-se de que uma política π é uma função que mapeia estados em ações. Estamos interessados principalmente em representações *parametrizadas* de π que têm muito menos parâmetros que a quantidade de estados existentes no espaço de estados (da mesma maneira que na seção anterior). Por exemplo, poderíamos representar π por uma coleção de funções *Q* parametrizadas, uma para cada ação, e executar a ação com o valor previsto mais alto:

$$\pi(s) = \max_a \hat{Q}_\theta(s, a) . \quad (21.14)$$

Cada função *Q* poderia ser uma função linear dos parâmetros θ, como na Equação 21.10, ou poderia ser uma função não linear como uma rede neural. A pesquisa de políticas ajustará então os parâmetros > para melhorar a política. Note que, se a política for representada por funções *Q*, a busca de políticas resulta em um processo que aprende funções *Q*. *Esse processo não é igual à aprendizagem Q!* Na aprendizagem *Q* com aproximação de função, o algoritmo encontra um valor de θ tal que $\hat{Q}_\theta$ está "próximo" de *Q**, a função *Q* ótima. Por outro lado, a busca de políticas encontra um valor de θ que resulta em bom desempenho; os valores encontrados pelos dois métodos podem diferir de maneira muito substancial (por exemplo, a função *Q* aproximada definida por $\hat{Q}_\theta(s, a) = Q^*(s, a)/10$ apresenta ótimo desempenho, mesmo se não estiver próxima de *Q**). Outro exemplo claro da diferença é o caso em que π(*s*) é calculado com o uso, digamos, da busca por antecipação de profundidade 10 com uma função utilidade aproximada $\hat{U}_\theta$. O valor de θ que permite um bom jogo pode estar longe de fazer $\hat{U}_\theta$ refletir a função utilidade verdadeira.

Um problema com as representações de políticas do tipo dado na Equação 21.14 é que a política é uma função *descontínua* dos parâmetros quando as ações são discretas (para um espaço de ação contínua, a política pode ser uma função suave dos parâmetros). Isto é, haverá valores de θ tais que uma mudança infinitesimal em θ fará a política passar de uma ação para outra. Isso significa que o valor da política também pode mudar descontinuamente, o que torna difícil a busca baseada em gradiente. Por essa razão, os métodos de busca de políticas frequentemente utilizam uma representação de **política estocástica** $\pi_\theta(s,a)$, que especifica a *probabilidade* de selecionar a ação *a* no estado *s*. Uma representação popular é a **função softmax**:

$$\pi_\theta(s, a) = e^{\hat{Q}_\theta(s,a)} / \sum_{a'} e^{\hat{Q}_\theta(s,a')} .$$

Softmax se torna quase determinística se uma ação é muito melhor que as outras, mas ela sempre fornece uma função diferenciável de θ; consequentemente, o valor da política (que depende de forma contínua das probabilidades de seleção de ações) é uma função diferenciável de θ. Softmax é uma

generalização da função logística (Seção 18.6.4) de múltiplas variáveis.

Agora, vamos examinar métodos para melhorar a política. Começaremos com o caso mais simples: uma política determinística e um ambiente determinístico. Seja $\rho(\theta)$ o **valor de política**, ou seja, a recompensa esperada quando π_θ é executado. Se podemos derivar uma expressão para $\rho(\theta)$ na forma fechada, temos um problema de otimização padrão, conforme descrito no Capítulo 4. Podemos seguir o vetor **gradiente de política** $\nabla_\theta \rho(\theta)$, desde que $\rho(\theta)$ seja diferenciável. Como alternativa, se $\rho(\theta)$ não estiver disponível na forma fechada, podemos avaliar π_θ simplesmente executando-o e observando a recompensa acumulada. Podemos seguir o **gradiente empírico** por subida de encosta, isto é, a avaliação da mudança na política para pequenos incrementos no valor de cada parâmetro. Com os gargalos habituais, esse processo convergirá para um local ótimo no espaço de políticas.

Quando o ambiente (ou a política) é estocástico, tudo fica mais difícil. Vamos supor que estejamos tentando executar a subida de encosta, que exige a comparação entre $\rho(\theta)$ e $\rho(\theta + \Delta\theta)$ para algum $\Delta\theta$ pequeno. O problema é que a recompensa total em cada trial pode variar amplamente e, assim, uma estimativa do valor da política a partir de um pequeno número de trials será pouco confiável; tentar comparar duas dessas estimativas será ainda mais incerto. Uma solução é simplesmente executar muitos trials, medindo a variância da amostra e usando-a para determinar que foram feitos suficientes trials para obter uma indicação confiável da direção da melhoria para $\rho(\theta)$. Infelizmente, isso é impraticável no caso de muitos problemas reais, nos quais cada trial pode ser dispendioso, demorado e talvez até mesmo perigoso.

Para o caso de uma política estocástica $\pi_\theta(s, a)$, é possível obter uma estimativa imparcial do gradiente em θ, $\nabla_\theta \rho(\theta)$, diretamente dos resultados de trials executados em θ. Por simplicidade, vamos derivar essa estimativa para o caso simples de um ambiente não sequencial em que a recompensa $R(a)$ é obtida imediatamente depois da execução da ação a no estado inicial s_0. Nesse caso, o valor da política é apenas o valor esperado da recompensa, e temos:

$$\nabla_\theta \rho(\theta) = \nabla_\theta \sum_a \pi_\theta(s_0, a) R(a) = \sum_a (\nabla_\theta \pi_\theta(s_0, a)) R(a) .$$

Agora, executamos um artifício simples para que esse somatório possa ser aproximado por amostras geradas a partir da distribuição de probabilidade definida por $\pi_\theta(s_0, a)$. Vamos supor que tenhamos N trials ao todo e que a ação executada no j-ésimo trial seja a_j. Então:

$$\nabla_\theta \rho(\theta) = \sum_a \pi_\theta(s_0, a) \cdot \frac{(\nabla_\theta \pi_\theta(s_0, a)) R(a)}{\pi_\theta(s_0, a)} \approx \frac{1}{N} \sum_{j=1}^{N} \frac{(\nabla_\theta \pi_\theta(s_0, a_j)) R(a_j)}{\pi_\theta(s_0, a_j)} .$$

Desse modo, o gradiente verdadeiro do valor da política é aproximado por uma soma de termos que envolvem o gradiente da probabilidade de seleção de ação em cada trial. Para o caso sequencial, isso é generalizado para:

$$\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{(\nabla_\theta \pi_\theta(s, a_j)) R_j(s)}{\pi_\theta(s, a_j)}$$

para cada estado s visitado, onde a_j é executada em s no j-ésimo trial, e $R_j(s)$ é a recompensa total recebida a partir do estado s em diante no j-ésimo trial. O algoritmo resultante é chamado REINFORCE (Williams, 1992); em geral, ele é muito mais efetivo que a subida de encosta usando grande quantidade de trials em cada valor de θ. No entanto, ainda é muito mais lento que o necessário.

Considere a tarefa a seguir: dados dois programas de vinte e um,[5] determinar qual deles é o melhor. Um modo de fazer isso é realizar cada jogo contra uma "banca" padrão por certo número de rodadas e depois medir seus respectivos ganhos. O problema com essa abordagem, como vimos, é que os ganhos de cada programa flutuam amplamente, dependendo do fato de ele receber cartas boas ou ruins. Uma solução óbvia é gerar certo número de rodadas com antecedência e *fazer com que cada programa jogue o mesmo conjunto de rodadas*. Desse modo, eliminamos o erro de medição devido a diferenças nas cartas recebidas. Essa ideia, chamada de **amostragem correlacionada**, forma a base de um algoritmo de busca de políticas chamado PEGASUS (Ng e Jordan, 2000). O algoritmo é aplicável a domínios para os quais um simulador está disponível, de forma que os resultados "aleatórios" das ações possam ser repetidos. O algoritmo funciona gerando com antecedência N sequências de números aleatórios, cada uma das quais podendo ser utilizada para executar um trial de qualquer política. A busca de políticas é executada avaliando-se cada política candidata com o *mesmo* conjunto de sequências aleatórias para determinar os resultados de ações. Podemos mostrar que o número de sequências aleatórias exigidas para assegurar que o valor de *toda* política é bem avaliado depende apenas da complexidade do espaço de políticas, e não da complexidade do domínio subjacente.

21.6 APLICAÇÕES DE APRENDIZAGEM POR REFORÇO

Passamos agora a exemplos de aplicações de aprendizagem por reforço em larga escala. Consideraremos aplicações em jogos, em que o modelo de transição é conhecido e o objetivo é aprender a função utilidade, e em robótica, onde o modelo geralmente é desconhecido.

21.6.1 Aplicativos para jogos

A primeira aplicação significativa de aprendizagem por reforço foi também o primeiro programa de aprendizagem significativo de qualquer tipo — o programa de jogo de damas escrito por Arthur Samuel (1959, 1967).

Samuel utilizou pela primeira vez uma função linear ponderada para a avaliação de posições, utilizando até 16 termos em qualquer instante. Ele aplicou uma versão da Equação 21.12 para atualizar os pesos. Havia algumas diferenças significativas, no entanto, entre o seu programa e os métodos atuais. Em primeiro lugar, ele atualizou os pesos usando a diferença entre o estado atual e o valor de cópia gerado através da previsão do efeito de escolha na árvore de busca. Isso funciona bem porque equivale à visualização do espaço de estados em uma granularidade diferente. A segunda diferença foi que o programa não usou qualquer recompensa observada! Ou seja, os valores dos

estados terminais alcançados eram ignorados. Isso significa que é teoricamente possível para o programa de Samuel não convergir ou convergir para uma estratégia destinada a perder em vez de ganhar. Ele conseguiu evitar esse destino insistindo que o peso de vantagem material deve sempre ser positivo. Notavelmente, isso foi suficiente para dirigir o programa em áreas de espaço de pesos correspondentes a bons jogos de damas.

O programa de gamão de Gerry Tesauro, TD-Gammon (1992), ilustra bem o potencial das técnicas de aprendizagem por reforço. Em trabalhos anteriores (Tesauro e Sejnowski, 1989), Tesauro tentou aprender uma representação de rede neural de $Q(s, a)$ diretamente de exemplos de movimentos rotulados com valores relativos por um especialista humano. Essa abordagem provou ser extremamente entediante para o especialista. Isso resultou em um programa, denominado NEUROGAMÃO, que era forte para os padrões do computador, mas não competitivo contra especialistas humanos. O projeto TD-Gammon foi uma tentativa de aprender a jogar sozinho. O único sinal de recompensa era dado ao final de cada jogo. A função de avaliação era representada por uma rede neural totalmente conectada com uma única camada oculta contendo 40 nós. Apenas pela aplicação repetida da Equação 21.12, o TD-Gammon aprendeu a jogar consideravelmente melhor do que o NEUROGAMÃO, embora a representação de entrada contivesse apenas a posição do tabuleiro cru sem as características calculadas. Isso levou cerca de 200 mil jogos de treinamento e duas semanas de tempo de computador. Embora possa parecer uma quantidade enorme de partidas, era apenas uma fração extremamente pequena do espaço de estado. Quando foram adicionadas características pré-calculadas à representação de entrada, uma rede com 80 nós ocultos foi capaz, depois de 300 mil jogos de treinamento, de alcançar um padrão de jogo comparável ao dos três melhores jogadores humanos em nível mundial. Kit Woolsey, um jogador de topo e analista, disse: “Não há dúvida em minha mente que sua decisão de posicionamento é muito melhor que a minha.”

21.6.2 Aplicação de controle de robô

A configuração do famoso problema de balanceamento do **carrinho com a vara**, também conhecido como pêndulo invertido, é mostrado na Figura 21.9. O problema é controlar a posição x do carrinho para que a vara fique aproximadamente na vertical ($\theta \approx \pi/2$), enquanto, como mostrado, permanece dentro dos limites da pista do carrinho. Vários milhares de artigos sobre aprendizagem por reforço e teoria de controle foram publicados sobre esse problema aparentemente simples. O problema do carrinho com vara difere dos problemas descritos anteriormente, em que as variáveis de estado x, θ, $\dot{x}$ e $\dot{\theta}$ são contínuas. As ações são geralmente discretas: sacudida para a esquerda ou para a direita, o chamado regime **de controle bangue-bangue**.

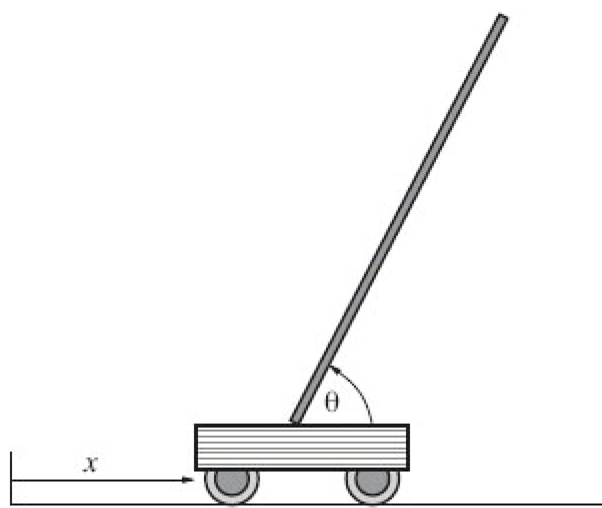


Figura 21.9 Configuração do problema de equilibrar uma vara comprida em cima de um carrinho em movimento. O carrinho pode ser sacudido para a esquerda ou para a direita por um controlador que observa x, θ, $\dot{x}$ e $\dot{\theta}$.

Os primeiros trabalhos sobre aprendizagem para esse problema foram realizados por Michie e Chambers (1968). Seu algoritmo BOXES foi capaz de equilibrar a vara por mais de uma hora depois de apenas 30 tentativas. Além disso, ao contrário de muitos sistemas subsequentes, BOXES foi implementado com um carrinho real e com a vara, e não uma simulação. O primeiro algoritmo discretizou o espaço de estados de quatro dimensões em caixas — daí o nome. Em seguida, realizou tentativas até que a vara caiu sobre o carro ou bateu ao final da pista. O reforço negativo foi associado com a ação final na caixa final e, então, propagado de volta através da sequência. Verificou-se que a discretização causou alguns problemas quando o dispositivo foi inicializado em posição diferente daquela usada no treinamento, sugerindo que a generalização não foi perfeita. Uma generalização melhorada e uma aprendizagem mais rápida podem ser obtidas utilizando um algoritmo que particiona o espaço de estados *de forma adaptativa* de acordo com a variação observada na recompensa ou usando uma aproximação de função de estado contínuo não linear, como uma rede neural. Hoje em dia, equilibrar uma *tripla* de pêndulo invertido é um exercício comum — um feito muito além da capacidade da maioria dos seres humanos.

Ainda mais impressionante é a aplicação de aprendizagem por reforço a um voo de helicóptero (Figura 21.10). Esse trabalho em geral tem utilizado a busca de políticas (Bagnell e Schneider, 2001), bem como o algoritmo PEGASUS com simulação baseada em um modelo de transição aprendido (Ng *et al*., 2004). No Capítulo 25 são apresentados mais detalhes.



Figura 21.10 Imagens superpostas de lapsos de tempo de um helicóptero autônomo executando uma manobra muito difícil de "círculo com nariz para dentro". O helicóptero está sob o controle de uma política desenvolvida pelo algoritmo de busca de políticas PEGASUS. Um modelo de simulador foi desenvolvido observando-se os efeitos de várias manipulações de controles no helicóptero real; em seguida, o algoritmo foi executado no modelo de simulador durante a noite. Uma variedade de controladores foi desenvolvida para manobras diferentes. Em todos os casos, o desempenho excedeu de longe o de um piloto especialista humano com a utilização de controle remoto (a imagem é cortesia de Andrew Ng).

21.7 RESUMO

Este capítulo examinou o problema da aprendizagem por reforço: como um agente pode se tornar proficiente em ambiente desconhecido, dadas apenas suas percepções e recompensas ocasionais. A aprendizagem por reforço pode ser visualizada como um microcosmo para o problema da IA inteira, mas é estudada em várias configurações simplificadas para facilitar o progresso. Os principais pontos são:

- O projeto de agente global dita o tipo de informações que devem ser aprendidas. Os três projetos principais que abordamos foram o projeto baseado em modelos, usando um modelo P e uma função utilidade U; o projeto livre de modelos, usando uma função de ação-valor Q; e o projeto reativo, usando uma política π.
- As utilidades podem ser aprendidas com o uso de três abordagens:

1. A **estimativa de utilidade direta** utiliza a recompensa total observada daí em diante para um dado estado como evidência direta para a aprendizagem de sua utilidade.
2. A **programação dinâmica adaptativa** (PDA) aprende um modelo e uma função de recompensa a partir de observações e depois utiliza a iteração de valor ou de política para obter as utilidades ou uma política ótima. A PDA faz uso ótimo das restrições locais sobre utilidades de estados impostas pela estrutura da vizinhança do ambiente.

3. Os métodos de **diferença temporal** (DT) atualizam estimativas de utilidade para corresponder às de estados sucessores. Eles podem ser visualizados como aproximações simples para a abordagem da PDA que pode aprender sem exigir um modelo de transição. Porém, o uso de um modelo aprendido para gerar pseudoexperiências pode resultar em aprendizagem mais rápida.

- Funções ação-valor, ou funções *Q*, podem ser aprendidas por uma abordagem de PDA ou de DT. Com DT, a aprendizagem *Q* não exige nenhum modelo, nem na fase de aprendizagem nem na de seleção de ações. Isso simplifica o problema de aprendizagem, mas restringe potencialmente a habilidade de aprender em ambientes complexos porque o agente não pode simular os resultados de cursos possíveis de ações.
- Quando o agente de aprendizagem é responsável pela seleção de ações enquanto aprende, ele deve assumir o compromisso entre o valor estimado dessas ações e o potencial para aprendizagem útil de novas informações. Uma solução exata do problema de exploração é inviável, mas algumas heurísticas simples realizam um trabalho razoável.
- Nos grandes espaços de estados, os algoritmos de aprendizagem por reforço devem usar representação funcional aproximada, a fim de generalizar sobre estados. O sinal de diferença temporal pode ser usado diretamente para atualizar parâmetros em representações como redes neurais.
- Os métodos de pesquisa de políticas operam diretamente em uma representação da política, tentando melhorá-la com base no desempenho observado. A variância no desempenho em domínio estocástico é um problema sério; para domínios simulados isso pode ser superado fixando-se a aleatoriedade com antecedência.

Devido a seu potencial para eliminar a codificação manual de estratégias de controle, a aprendizagem por reforço continua a ser uma das áreas mais ativas da pesquisa de aprendizagem de máquina. As aplicações em robótica prometem ser particularmente valiosas; elas exigirão métodos para manipulação de ambientes *contínuos*, de dimensões elevadas e parcialmente observáveis, nos quais os comportamentos bem-sucedidos podem consistir em milhares ou até milhões de ações primitivas.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Turing (1948, 1950) propôs a abordagem de aprendizagem por reforço, embora ele não estivesse seguro de sua efetividade, escrevendo: "O uso de penalidades e recompensas pode ser, no máximo, uma parte do processo de ensino." O trabalho de Arthur Samuel (1959) foi provavelmente a pesquisa de aprendizagem de máquina mais antiga a obter sucesso. Embora fosse informal e tivesse várias falhas, esse trabalho continha a maior parte das ideias modernas em aprendizagem por reforço, inclusive a diferenciação temporal e a aproximação de funções. Por volta da mesma época, os pesquisadores em teoria de controle adaptativo (Widrow e Hoff, 1960), baseando-se no trabalho de Hebb (1949), estavam treinando redes simples com o emprego da regra delta (essa conexão inicial entre redes neurais e aprendizagem por reforço pode ter sido a origem do persistente equívoco de percepção segundo o qual este último é um subcampo do outra). O trabalho da vara no carrinho

realizado por Michie e Chambers (1968) também pode ser visto como um método de aprendizagem por reforço com aproximador de função. A literatura na área de psicologia sobre aprendizagem por reforço é muito mais antiga; Hilgard e Bower (1975) apresentam um bom levantamento dela. A evidência direta da operação de aprendizagem por reforço em animais foi fornecida pelas investigações no comportamento exploratório das abelhas; existe clara correlação neural do sinal de recompensa, sob a forma de um grande mapeamento de neurônios desde os sensores de influxo de néctar diretamente para o córtex motor (Montague *et al*., 1995). A pesquisa com utilização de registros de uma única célula sugere que o sistema de dopamina no cérebro de primatas implementa algo semelhante à aprendizagem de funções de valor (Schultz *et al*., 1997). O texto de neurociência por Dayan e Abbott (2001) descreve possíveis implementações neurais de aprendizagem de diferença temporal, enquanto Dayan e Niv (2008) pesquisaram a evidência mais recente a partir de experimentos neurocientíficos e comportamentais.

A conexão entre aprendizagem por reforço e processos de decisão de Markov foi realizada primeiro por Werbos (1977), mas o desenvolvimento da aprendizagem por reforço em IA teve origem no trabalho realizado na Universidade de Massachusetts no início da década de 1980 (Barto *et al*., 1981). O ensaio de Sutton (1988) fornece uma boa visão geral histórica. A Equação 21.3 deste capítulo é um caso especial para $\lambda = 0$ do algoritmo geral DT(λ) de Sutton. O DT(λ) atualiza os valores de utilidade de todos os estados em uma sequência, a fim de preparar o caminho para cada transição em uma proporção que cai como λ^t para etapas de t estados anteriores.

DT(λ) é idêntico à regra de Widrow-Hoff ou regra delta. Boyan (2002), fundamentando-se no trabalho de Bradtke e Barto (1996), argumenta que DT(λ) e algoritmos inter-relacionados fazem uso ineficiente das experiências; em essência, eles são algoritmos de regressão on-line que convergem muito mais lentamente que a regressão off-line. Seu LSTD (diferenciação temporal baseada em mínimos quadrados) é um algoritmo on-line para aprendizagem com reforço passivo que fornece os mesmos resultados da regressão off-line. A iteração de política baseada nos mínimos quadrados ou IPMQ (Lagoudakis e Parr, 2003), combina essa ideia com o algoritmo de iteração de política, produzindo um algoritmo robusto, estatisticamente eficiente, livre de modelo para as políticas de aprendizagem.

A combinação da aprendizagem de diferença temporal com a geração baseada em modelos de experiências simuladas foi proposta na arquitetura DYNA de Sutton (Sutton, 1990). A ideia de varredura priorizada foi introduzida independentemente por Moore e Atkeson (1993) e por Peng e Williams (1993). A aprendizagem Q foi desenvolvida na tese de doutorado de Watkins (1989), enquanto SARSA apareceu em um relatório técnico por Rummery e Niranjan (1994).

Os problemas do bandido, que modelam o problema de exploração para decisões não sequenciais, são analisados em profundidade por Berry e Fristedt (1985). Estratégias de exploração ótimas para várias configurações podem ser obtidas com o emprego da técnica denominada **índices de Gittins** (Gittins, 1989). Diversos métodos de exploração para problemas de decisão sequenciais são examinados por Barto *et al*. (1995). Kearns e Singh (1998) e Brafman e Tennenholtz (2000) descrevem algoritmos que exploram ambientes desconhecidos e oferecem a garantia de convergir sobre políticas quase ótimas em tempo polinomial. A aprendizagem bayesiana por reforço (Dearden *et al*., 1998, 1999) fornece um outro ângulo da abordagem, tanto para o modelo da incerteza como de exploração.

A aproximação de funções em aprendizagem por reforço surgiu com o trabalho de Samuel, que empregou funções de avaliação lineares e não lineares, e também usou métodos de seleção de características para reduzir o espaço de características. Os métodos mais antigos incluem o **CMAC** (*Cerebellar Model Articulation Controller*) (Albus, 1975), que é essencialmente uma soma de funções superpostas de núcleo local, e as redes neurais associativas de Barto *et al*. (1983). As redes neurais atualmente são a forma mais popular de aproximador de função. A aplicação mais conhecida é o TD-Gammon (Tesauro, 1992, 1995), que foi examinado no capítulo. Um problema significativo exibido por agentes de aprendizagem de DT baseados em redes neurais é que eles tendem a esquecer experiências mais antigas, especialmente as que surgem em partes do espaço de estados que são evitadas uma vez que a competência é alcançada. Isso pode resultar em fracasso catastrófico se tais circunstâncias reaparecerem. A aproximação de função baseada em **aprendizagem baseada em instâncias** pode evitar esse problema (Ormoneit e Sen, 2002; Forbes, 2002).

A convergência de algoritmos de aprendizagem por reforço usando a aproximação de função é um assunto extremamente técnico. Os resultados para aprendizagem de DT foram progressivamente fortalecidos para o caso de aproximadores de funções lineares (Sutton, 1988; Dayan, 1992; Tsitsiklis e Van Roy, 1997), mas vários exemplos de divergência foram apresentados para funções não lineares (veja em Tsitsiklis e Van Roy, 1997, uma discussão). Papavassiliou e Russell (1999) descrevem um novo tipo de aprendizagem por reforço que converge com qualquer forma de aproximador de função, desde que uma aproximação de melhor ajuste possa ser encontrada para os dados observados.

Os métodos de busca de políticas foram trazidos à luz por Williams (1992), que desenvolveu a família de algoritmos REINFORCE. O trabalho posterior de Marbach e Tsitsiklis (1998), de Sutton *et al*. (2000) e de Baxter e Bartlett (2000) fortaleceu e generalizou os resultados de convergência para busca de políticas. O método de amostragem correlacionada para comparar configurações diferentes de um sistema foi descrito formalmente por Kahn e Marshall (1953), mas parece que era conhecido bem antes disso. Seu uso em aprendizagem por reforço é devido a Van Roy (1998) e Ng e Jordan (2000); o último artigo também apresentou o algoritmo PEGASUS e provou suas propriedades formais.

Como mencionamos no capítulo, o desempenho de uma política *estocástica* é uma função contínua de seus parâmetros, o que ajuda no caso de métodos de busca baseados em gradiente. Esse não é o único benefício: Jaakkola *et al*. (1995) argumentam que as políticas estocásticas realmente funcionam melhor que as políticas determinísticas em ambientes parcialmente observáveis se ambas estiverem limitadas a agir com base na percepção atual (uma razão para isso é que a política estocástica tem menor probabilidade de ficar “paralisada” devido a algum impedimento não previsto). No Capítulo 17, assinalamos que as políticas ótimas em MDPs parcialmente observáveis são funções determinísticas do *estado de crença*, e não da percepção atual, e assim seriam de esperar resultados ainda melhores controlando-se o estado de crença com o uso dos métodos de **filtragem** do Capítulo 15. Infelizmente, o espaço de estados de crença é altamente dimensional e contínuo, e ainda não foram desenvolvidos algoritmos efetivos para aprendizagem por reforço com estados de crença.

Os ambientes reais também exibem enorme complexidade em termos do número de ações primitivas exigidas para se alcançar uma recompensa significativa. Por exemplo, um robô que joga futebol poderia fazer cem mil movimentos individuais das pernas antes de marcar um gol. Um método

comum, usado originalmente no treinamento de animais, é chamado **modelagem de recompensa**. Envolve oferecer ao agente recompensas adicionais, chamadas de **pseudorrecompensas**, por "fazer progressos". Por exemplo, no futebol, a verdadeira recompensa é marcar um gol, mas poderiam ser oferecidas pseudorrecompensas ao fazer contato com a bola ou ao chutá-la em direção ao gol. Tais recompensas podem acelerar enormemente a aprendizagem, e é muito simples fornecê-las, mas existe o risco de que o agente aprenda a maximizar as pseudorrecompensas em lugar das recompensas verdadeiras; por exemplo, ficar próximo à bola e "vibrar" provoca muitos contatos com a bola. Ng *et al*. (1999) mostram que o agente ainda aprenderá a política ótima desde que a pseudorrecompensa $F(s, a, s')$ satisfaça $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$, onde Φ é uma função arbitrária do estado. Φ pode ser construída de modo a refletir aspectos desejáveis do estado, como a realização de subobjetivos ou a distância até um estado objetivo.

A geração de comportamentos complexos também pode ser facilitada por métodos de **aprendizagem por reforço hierárquico**, que tentam resolver problemas em vários níveis de abstração — de modo muito semelhante aos métodos de **planejamento de HTN** do Capítulo 11. Por exemplo, "marcar um gol" pode ser desmembrado em "obter a posse da bola", "driblar em direção à meta" e "chutar", e cada uma dessas ações pode ser dividida mais ainda em comportamentos motores de nível mais baixo. O resultado fundamental nessa área se deve a Forestier e Varaiya (1978), que provaram que comportamentos de nível mais baixo de complexidade arbitrária podem ser tratados simplesmente como ações primitivas (embora sejam ações que podem ter durações variadas de tempo) do ponto de vista do comportamento de nível mais alto que os invoca. Abordagens atuais de Parr e Russell (1998), Dietterich (2000), Sutton *et al*. (2000), Andre e Russell (2002) fundamentam seu trabalho nesse resultado para desenvolver métodos que fornecem a um agente um **programa parcial** que restringe o comportamento do agente a ter uma estrutura hierárquica específica. A linguagem de programação parcial para os programas de agente estende uma linguagem de programação normal adicionando primitivas para escolhas não especificadas, que devem ser preenchidas pela aprendizagem. A aprendizagem por reforço é então aplicada para aprender o melhor comportamento consistente com o programa parcial. A combinação de aproximação de função, modelagem e aprendizagem por reforço hierárquico provou resolver problemas em grande escala — por exemplo, políticas que executam 10^4 etapas em espaços de estado de 10^{100} estados com fatores de ramificação de 10^{30} (Marthi *et al*., 2005). Um resultado-chave (Dietterich, 2000) é que a estrutura hierárquica fornece uma *decomposição aditiva* natural da função utilidade global em termos que dependem de pequenos subconjuntos de variáveis que definem o espaço de estados. Isso é um pouco análogo aos teoremas de representação subjacentes à concisão das redes de Bayes (Capítulo 14).

O tema da aprendizagem por reforço distribuído e multiagente não foi abordado no capítulo, mas é de grande interesse atual. Em AR distribuído, o objetivo é conceber métodos pelos quais agentes múltiplos coordenados aprendem a otimizar uma função utilidade comum. Por exemplo, podemos conceber métodos pelos quais **subagentes** separados para navegação de robôs e de fuga de obstáculos por robôs poderiam alcançar cooperativamente um sistema de controle combinado que fosse globalmente melhor? Obtiveram-se alguns resultados básicos nessa direção (Guestrin *et al*., 2002;. Russell e Zimdars, 2003). A ideia básica é que cada subagente aprenda sua própria função Q de seu próprio fluxo de recompensas. Por exemplo, um componente de navegação de robôs pode receber recompensas por evoluir em direção à meta, enquanto o componente de fuga de obstáculos

recebe recompensas negativas em cada colisão. Cada decisão global maximiza a soma das funções Q e todo o processo converge para soluções globalmente ótimas.

O multiagente AR se distingue do distribuído AR pela presença de agentes que não podem coordenar suas ações (exceto por atos comunicativos explícitos) e que não podem compartilhar a mesma função utilidade. Assim, o multiagente AR lida com problemas de jogo teórico sequencial ou **jogos de Markov**, conforme definido no Capítulo 17. A exigência consequente de políticas randomizadas não é uma complicação importante, como vimos anteriormente. O que causa problemas é o fato de que, enquanto um agente está aprendendo a derrotar a política de seu adversário, o adversário está mudando sua política para derrotar o agente. Assim, o ambiente é **não estacionário**. Littman (1994) observou essa dificuldade ao apresentar os primeiros algoritmos AR para os jogos de Markov de soma zero. Hu e Wellman (2003) apresentam um algoritmo de aprendizagem geral Q para jogos de soma que convergem quando o equilíbrio de Nash é único; quando houver vários equilíbrios, a noção de convergência não é tão fácil de definir (Shoham *et al*., 2004).

Às vezes, a função de recompensa não é fácil de definir. Considere a tarefa de dirigir um carro. Há estados extremos (como bater o carro) que certamente deveriam ter uma grande penalidade. Mas, além disso, é difícil ser preciso sobre a função de recompensa. No entanto, é suficientemente fácil para um ser humano dirigir por um tempo e depois dizer a um robô "faça-o assim". O robô então tem a tarefa de **aprendizado da aprendizagem**; aprender com um exemplo de tarefa bem-feita, sem recompensas explícitas. Ng *et al*. (2004) e Coates *et al.* (2009) mostram como essa técnica funciona para aprender a pilotar um helicóptero; veja a Figura 25.25 para um exemplo das acrobacias que a política resultante é capaz de fazer. Russell (1998) descreve a tarefa de **aprendizagem por reforço inversa** imaginando qual deve ser a função de recompensa a partir de um caminho de exemplo através desse espaço de estados. Isso é útil como parte do aprendizado da aprendizagem ou como parte de fazer ciência — podemos entender um animal ou robô retrocedendo a partir do que ele faz para o que deva ser a sua função de recompensa.

Este capítulo abordou apenas os estados — todos os agentes sabem que um estado é o conjunto de ações disponíveis e utilidades dos estados resultantes (ou dos pares estado-ação). Mas também é possível aplicar a aprendizagem por reforço em representações para representações estruturadas em vez de atômicas, o que é chamado de **aprendizagem por reforço relacional** (Tadepalli *et al.*, 2004).

A pesquisa de Kaelbling *et al*. (1996) fornece um bom ponto de entrada para a literatura. O texto de Sutton e Barto (1998), dois pioneiros do campo, se concentra em arquiteturas e algoritmos, mostrando como a aprendizagem por reforço reúne as ideias de aprendizagem, planejamento e ação. O trabalho um pouco mais técnico desenvolvido por Bertsekas e Tsitsiklis (1996) oferece uma base rigorosa para a teoria de programação dinâmica e convergência estocástica. Ensaios e artigos sobre aprendizagem por reforço são publicados com frequência em *Machine Learning*, no *Journal of Machine Learning Research* e nas reuniões das International Conferences on Machine Learning e Neural Information Processing Systems.

EXERCÍCIOS

21.1 Implemente um agente de aprendizagem passivo em um ambiente simples, como o mundo

4 × 3. Para o caso de um modelo de ambiente inicialmente desconhecido, compare o desempenho de aprendizagem dos algoritmos de estimativa de utilidade direta, DT e PDA. Faça a comparação para a política ótima e para várias políticas aleatórias. Para quais delas as estimativas de utilidade convergem mais rapidamente? O que acontece quando o tamanho do ambiente é ampliado? (Tente ambientes com e sem obstáculos.)

21.2 O Capítulo 17 definiu uma **política própria** para um MDP como uma política que garante alcançar o estado terminal. Mostre que para um agente PDA passivo é possível aprender um modelo de transição para o qual sua política π seja própria para o MDP verdadeiro. Com tais modelos a etapa de AVALIAÇÃO-DE-POLÍTICA pode falhar se $\gamma = 1$. Mostre que esse problema não irá surgir se a AVALIAÇÃO-DE-POLÍTICA for aplicada ao modelo aprendido apenas ao final de um trial.

21.3 Começando com o agente de PDA passivo, modifique-o para usar um algoritmo de PDA aproximado, conforme discutimos no texto. Faça isso em duas etapas:

a. Implemente uma fila de prioridades para ajustes das estimativas de utilidade. Sempre que um estado é ajustado, todos os seus predecessores também se tornam candidatos ao ajuste e devem ser adicionados à fila. A fila é inicializada com o estado a partir do qual ocorreu a transição mais recente. Permita apenas um número fixo de ajustes.

b. Faça experiências com várias heurísticas para ordenar a fila de prioridades, examinando seu efeito sobre as taxas de aprendizagem e o tempo de computação.

21.4 Escreva as equações de atualização de parâmetros para a aprendizagem de DT com:

$$\hat{U}(x,y) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 \sqrt{(x-x_g)^2 + (y-y_g)^2}\,.$$

21.5 Implemente um agente de aprendizagem por reforço de exploração que utiliza estimativa de utilidade direta. Crie duas versões — uma com representação tabular e outra usando o aproximador de função da Equação 21.10. Compare seu desempenho em três ambientes:

a. O mundo 4 × 3 descrito no capítulo.

b. Um mundo 10 × 10 sem obstáculos e uma recompensa +1 em (10,10).

c. Um mundo 10 × 10 sem obstáculos e uma recompensa +1 em (5,5).

21.6 21.6 Conceba características adequadas para o reforço de aprendizagem em mundos de grade estocásticos (generalizações do mundo 4 x 3) que contenha obstáculos e estados terminais múltiplos com recompensas de +1ou -1.

21.7 Estenda o ambiente de jogo padrão (Capítulo 5) para incorporar um sinal de recompensa. Coloque dois agentes de aprendizagem por reforço no ambiente (eles podem, é claro, compartilhar o programa de agente) e fazê-los jogar uns contra os outros. Aplique a regra de atualização de DT generalizada (Equação 21.12) para atualizar a função de avaliação. Você pode começar com uma função de avaliação ponderada linear simples e um jogo simples, como o jogo da velha.

21.8 Calcule a função utilidade verdadeira e a melhor aproximação linear em x e y (como na

Equação 21.10) para os ambientes a seguir:

a. Um mundo 10 × 10 com um único estado terminal +1 em (10,10).

b. Como em (a), mas adicione um estado terminal –1 em (10,1).

c. Como em (b), mas adicione obstáculos em 10 quadrados selecionados ao acaso.

d. Como em (b), mas inclua uma parede indo de (5,2) até (5,9).

e. Como em (a), mas com o estado terminal em (5,5).

As ações são movimentos determinísticos nas quatro direções. Em cada caso, compare os resultados usando representações tridimensionais. Para cada ambiente, proponha características adicionais (além de x e y) que melhorariam a aproximação e mostre os resultados.

21.9 Implemente os algoritmos REINFORCE e PEGASUS e aplique-os ao mundo de 4 × 3, utilizando uma família de políticas de sua própria escolha. Comente os resultados.

21.10 A aprendizagem por reforço é um modelo abstrato adequado de evolução? Que relação existe, se houver, entre os sinais de recompensa fisicamente programada e a aptidão evolutiva?

---

[1] As condições técnicas foram dadas no Capítulo 18.6. Na Figura 21.5 temos apenas $a(n) = 60/59+n)$, que satisfaz as condições.

[2] Note a analogia direta com a teoria de valor da informação que estudamos no Capítulo 16.

[3] Sabemos que a função utilidade exata pode ser representada em uma página ou duas de Lisp, Java ou C++. Isto é, ela pode ser representada por um programa que resolve o jogo de forma exata toda vez que é chamado. Estamos interessados apenas em aproximadores de funções que utilizam uma quantidade *razoável* de computação. De fato, talvez fosse melhor aprender um aproximador de função muito simples e combiná-lo com certa quantidade de busca por antecipação. Os compromissos envolvidos ainda não estão bem compreendidos nos dias de hoje.

[4] A definição de distância entre funções utilidade é bastante técnica; veja Tsitsiklis e Van Roy (1997).

[5] Também conhecido como 21, pontão ou blackjack.

PARTE VI

Comunicação, percepção e ação

CAPÍTULO 22

Processamento de linguagem natural

Em que vemos como fazer uso do conhecimento abundante que se expressa na linguagem natural.

O *Homo sapiens* é separado de outras espécies pela capacidade da linguagem. Algo em torno de 100.000 anos atrás, os seres humanos aprenderam a falar e cerca de 7.000 anos atrás aprenderam a escrever. Embora os chimpanzés, golfinhos e outros animais mostrem vocabulário com centenas de sinais, só os seres humanos podem comunicar de forma confiável um número ilimitado de mensagens qualitativamente diferentes sobre qualquer tema utilizando sinais discretos.

É claro, existem outros atributos que são exclusivamente humanos: nenhuma outra espécie usa roupas, cria arte com representações ou vê três horas de televisão por dia. Mas, quando Alan Turing propôs o seu Teste (veja Seção 1.1.1), ele o baseou em linguagem e não em arte ou TV. Há duas razões principais pelas quais queremos que nossos agentes de computador sejam capazes de processar linguagens naturais: primeiro, para se comunicar com os seres humanos, um tema que veremos no Capítulo 23 e, segundo, para adquirir informação a partir da linguagem escrita, o foco deste capítulo.

Há mais de um trilhão de páginas de informações na Web, quase todas elas em linguagem natural. Um agente que deseja **adquirir conhecimento** precisa entender (pelo menos parcialmente) a ambígua e confusa linguagem que os seres humanos usam. Examinaremos o problema do ponto de vista de tarefas de busca de informações específicas: classificação de texto, recuperação de informação e extração de informação. Um fator comum na abordagem dessas tarefas é o uso de **modelos de linguagem**: modelos que preveem a distribuição de probabilidade das expressões de linguagem.

22.1 MODELOS DE LINGUAGEM

As linguagens formais, tais como as linguagens de programação Java ou Python, têm modelos de linguagem precisamente definidos. Uma **linguagem** pode ser definida como um conjunto de sequências; "`print (2 + 2)`" é um programa válido na linguagem Python, enquanto "`2) + (2 print`" não é. Uma vez que há um número infinito de programas válidos, eles não podem ser enumerados; em vez disso são especificados por um conjunto de regras chamado **gramática**. As linguagens formais também têm regras que definem o significado ou a **semântica** de um programa,

por exemplo, as regras dizem que o “significado” de “`2 + 2`” é 4, e o significado de “`1/0`” é que será sinalizado erro.

As linguagens naturais, tais como o inglês ou o espanhol, não podem ser caracterizadas como um conjunto de sentenças definitivas. Todos concordam que “Não ser convidado é triste” é uma sentença em português, mas as pessoas discordam sobre a gramaticalidade de “Não ser convidado é triste”. Portanto, é mais frutífero definir um modelo de linguagem natural, como uma distribuição de probabilidade sobre sentenças em vez de um conjunto definitivo. Ou seja, em vez de perguntar se uma sequência de *palavras* é ou não membro do conjunto que define o idioma, perguntamos por $P(S = palavras)$ — qual a probabilidade de que uma sentença aleatória seria *palavras*.

As linguagens naturais também são **ambíguas**. “Ele viu o banco” pode tanto significar que ele viu uma peça de mobiliário, ou que viu uma instituição financeira. Assim, novamente, não podemos falar de um único significado para uma sentença, mas de uma distribuição de probabilidade sobre significados possíveis.

Finalmente, as linguagens naturais são difíceis de lidar porque são muito grandes e em constante mutação. Assim, os modelos de nossa língua são, na melhor das hipóteses, uma aproximação. Começamos com as aproximações mais simples possíveis e continuaremos a partir daí.

22.1.1 Modelos de caracteres de *N*-grama

Em última análise, um texto escrito é composto por **caracteres** — letras, dígitos, pontuação e espaços (e caracteres mais exóticos em alguns outros idiomas). Assim, um dos modelos de linguagem mais simples é uma distribuição de probabilidade sobre sequências de caracteres. Como no Capítulo 15, escrevemos $P(c_{1:n})$ para a probabilidade de uma sequência de N caracteres, de c_1 até de c_n. Em uma coleção da Web, $P(“a”) = 0{,}027$ e $P(“zgq”) = 0{,}000000002$. Uma sequência de símbolos escritos de comprimento n é chamado de n-grama (da raiz grega para escrita ou letras), com o caso especial de 1-grama para “unigrama”, 2-gramas para “bigrama” e 3-gramas pra “trigrama”. Um modelo de distribuição de probabilidade de n sequências de letras é então chamado de modelo de ***n*-grama** (mas, cuidado: podemos ter modelos de n-gramas com sequências de palavras, sílabas ou outras unidades, não apenas de caracteres).

Um modelo de n-grama é definido como uma **cadeia de Markov** de ordem $n - 1$. Lembre-se que em uma cadeia de Markov, há a probabilidade de o caractere ci depender apenas dos caracteres imediatamente anteriores e não de qualquer outro caractere. Assim, em um modelo de trigrama (cadeia de Markov de ordem 2) temos

$$P(c_i \mid c_{1:i-1}) = P(c_i \mid c_{i-2:i-1}) \, .$$

Podemos definir a probabilidade de uma sequência de caracteres P(c1: N) sob o modelo do trigrama primeiro por fatoração com a regra de cadeia e, em seguida, utilizando a hipótese de Markov:

$$P(c_{1:N}) = \prod_{i=1}^{N} P(c_i \mid c_{1:i-1}) = \prod_{i=1}^{N} P(c_i \mid c_{i-2:i-1})\,.$$

Para um modelo de caracteres de trigrama em uma linguagem com 100 caracteres, $\mathbf{P}(C_i \mid C_{i-2:i-1})$ tem um milhão de entradas e pode ser estimado com precisão, contando as sequências de caracteres em um corpo de texto de 10 milhões de caracteres ou mais. Denominaremos **corpus** (plural *corpora)* um corpo de texto, palavra latina para *corpo*.

O que podemos fazer com os modelos de caracteres de *n*-gramas? Uma tarefa para a qual eles estão bem adaptados é a **identificação de linguagem**: dado um texto, determine qual linguagem natural está escrita nele. Essa é uma tarefa relativamente fácil, mesmo com textos curtos, como "*Hello, world*" ou "*Wie geht es dir*"; é fácil identificar o primeiro como inglês e o segundo como alemão. Os sistemas de computador identificam os idiomas com mais de 99% de precisão; ocasionalmente é confuso identificar idiomas que estão intimamente relacionados, tais como sueco e norueguês.

Uma abordagem para identificação de idioma é primeiro construir um modelo de caracteres de trigrama de cada idioma candidato, $P(c_i \mid c_{i-2:i-1},\ell)$, onde a variável ℓ estende-se entre os idiomas. Para cada ℓ é construído um modelo através da contagem de trigramas em um *corpus* dessa língua (são necessários cerca de 100.000 caracteres de cada língua). Isso nos dá um modelo de **P**(*Texto* | *Linguagem*), mas, dado o texto, queremos selecionar o idioma mais provável, então aplicamos a regra de Bayes seguida pela suposição de Markov para obter o idioma mais provável:

$$\begin{aligned}\ell^* &= \operatorname*{argmax}_{\ell} P(\ell \mid c_{1:N}) \\ &= \operatorname*{argmax}_{\ell} P(\ell) P(c_{1:N} \mid \ell) \\ &= \operatorname*{argmax}_{\ell} P(\ell) \prod_{i=1}^{N} P(c_i \mid c_{i-2:i-1}, \ell)\,.\end{aligned}$$

O modelo de trigrama pode ser aprendido a partir de um *corpus*, mas, e sobre a probabilidade anterior P(l)? Podemos ter uma estimativa desses valores; por exemplo, se estivermos selecionando uma página aleatória da Web, sabemos que o inglês é o idioma mais provável e que a probabilidade de macedônio é inferior a 1%. O número exato que selecionamos para essas medidas *a priori* não é crítico porque o modelo de trigrama normalmente seleciona uma linguagem que será várias ordens de magnitude mais provável que qualquer outra.

Outras tarefas para modelos de caracteres incluem a correção de ortografia, classificação de gênero e o chamado reconhecimento de entidade. A classificação de gênero significa decidir se um texto é uma nova história, um documento legal, um artigo científico etc. Embora muitas características ajudem a fazer essa classificação, o resultado da pontuação e outros caracteres *n*-grama característicos podem contribuir muito (Kessler *et al*., 1997). O objetivo da tarefa denominada reconhecimento de entidade é o de encontrar os nomes das coisas em um documento e decidir a qual classe pertencem. Por exemplo, no texto "Foi prescrito Aciphex para o Sr. Sopersteen", devemos reconhecer que "o Sr. Sopersteen" é o nome de uma pessoa e "Aciphex" é o nome de um medicamento. Os modelos de nível de caracteres são bons para essa tarefa, pois podem

associar a sequência de caracteres "ex␣" ("ex" seguido por um espaço) com um nome de medicamento e "steen␣" com um nome de pessoa e, assim, identificar palavras que nunca viram antes.

22.1.2 Alisamento de modelos *n*-grama

A principal complicação de modelos *n*-grama é que o *corpus* de treinamento fornece apenas uma estimativa da distribuição de probabilidade verdadeira. Para sequências de caracteres comuns, como "␣th" qualquer *corpus* em inglês dará uma boa estimativa: cerca de 1,5% de todos os trigramas. Por outro lado, "␣th" não é muito comum — nenhuma palavra do dicionário começa com ht. É provável que a sequência tivesse uma contagem de zero em um *corpus* de treinamento de padrão inglês. Isso significa que se deve atribuir P("␣th") = 0? Se fizermos isso, o texto "The program issues an http request" teria uma probabilidade de inglês zero, o que parece errado. Temos um problema de generalização: queremos que nossos modelos de linguagem generalizem bem os textos que ainda não tenham visto. Apenas porque nunca vimos "␣http" antes não significa que nosso modelo deva afirmar que é impossível. Assim, vamos ajustar o nosso modelo de linguagem de forma que será atribuída uma pequena probabilidade diferente de zero às sequências que tiverem resultado zero no *corpus* de treinamento (e os outros resultados serão ajustados ligeiramente para baixo de modo que a probabilidade ainda some 1). O processo de ajustar o resultado da probabilidade de baixa frequência é chamado de **alisamento**.

O tipo mais simples de alisamento foi sugerido por Pierre-Simon Laplace, no século XVIII: ele disse que, na falta de mais informações, se uma variável aleatória booleana X for falsa em todas as n observações até agora, a estimativa de $P(X = verdadeiro)$ deve ser $1/(n + 2)$. Isto é, supõe-se que, com mais duas tentativas, exista a chance de ser um verdadeiro e outro falso. O alisamento de Laplace (também chamado de alisamento exponencial) é um passo na direção certa, mas tem desempenho relativamente fraco. Uma abordagem melhor é o **modelo de recuo** (**backoff**), em que começamos por estimar o resultado de *n*-grama, mas para qualquer sequência particular que tenha esultado baixo (ou zero), recuamos para $(n - 1)$ gramas. A **interpolação linear com alisamento** é um modelo de recuo que combina modelos trigrama, bigrama e unigrama por interpolação linear. Ela define a estimativa de probabilidade como

$$\widehat{P}(c_i|c_{i-2:i-1}) = \lambda_3 P(c_i|c_{i-2:i-1}) + \lambda_2 P(c_i|c_{i-1}) + \lambda_1 P(c_i) \,,$$

onde $\lambda_3 + \lambda_2 + \lambda_1 = 1$. Os valores do parâmetro λ_i podem ser fixados ou treinados com um algoritmo de maximização de expectativa (EM). Também é possível ter os valores de λ_i dependentes do resultado: se tivermos um resultado de trigrama alto, atribuímos a ele um peso relativamente maior; se for apenas um resultado baixo, colocamos mais peso nos modelos de bigrama e unigrama. Um grupo de pesquisadores desenvolveu modelos de alisamento cada vez mais sofisticado, enquanto o outro grupo sugeriu reunir um *corpus* maior de modo que mesmo os modelos de alisamento simples funcionassem bem. Ambos estão tentando atingir a mesma meta: reduzir a variância no modelo de linguagem.

Uma complicação: observe que a expressão $P(c_i \mid c_{i-2:i-1})$ pede por $P(c_1 \mid c_{-1:0})$ quando $i = 1$, mas

não há caracteres antes de c_1. Podemos introduzir caracteres artificiais, por exemplo, definir c0 como um caractere de espaço ou um caractere de "início de texto" especial. Ou podemos recair sobre os modelos de ordem inferior de Markov, com efeito definindo $c_{-1:0}$ como sequência vazia e, portanto, $P(c_1 \mid c_{-1:0}) = P(c_1)$.

22.1.3 Avaliação de modelo

Com tantos modelos possíveis de *n*-grama, bigrama, trigrama, com interpolação de alisamento com valores diferentes de λ etc., como saber que modelo escolher? Podemos avaliar um modelo com validação cruzada. Dividir o *corpus* em um *corpus* de treinamento e um *corpus* de validação. Determinar o modelo de treinamento a partir dos dados de treinamento. Em seguida, avaliar o modelo pela validação do *corpus*.

A avaliação pode ser uma métrica de tarefa específica, tal como medir a precisão da identificação da linguagem. Alternativamente, podemos ter um modelo de tarefas independentes de qualidade da linguagem: calcular a probabilidade atribuída ao *corpus* de validação pelo modelo; quanto maior for a probabilidade, melhor. Essa métrica é inconveniente porque a probabilidade de um grande *corpus* será um número muito pequeno, e um subfluxo de ponto flutuante torna-se um problema. Uma forma diferente de descrever a probabilidade de uma sequência é com uma medida chamada **perplexidade**, definida como

$$Perplexidade\,(c_{1:N}) = P(c_{1:N})^{-\frac{1}{N}}\ .$$

A perplexidade pode ser imaginada como o inverso da probabilidade, normalizada pelo comprimento da sequência. Pode também ser imaginada como o fator de ramificação médio ponderado de um modelo. Suponha que haja 100 caracteres em nossa língua, e nosso modelo informa que todos eles têm a mesma probabilidade. Em seguida, para uma sequência de qualquer tamanho, a perplexidade será 100. Se alguns caracteres forem mais prováveis do que outros, e o modelo reflete isso, o modelo terá uma perplexidade menor que 100.

22.1.4 Modelos de palavra de *N*-grama

Agora voltamos aos modelos de *n*-grama de palavras em vez de caracteres. Todos os mecanismos aplicam-se igualmente aos modelos de palavra e de caracteres. A principal diferença é que o **vocabulário** — o conjunto de símbolos que compõem o *corpus* e o modelo — é maior. Há apenas cerca de 100 caracteres na maioria dos idiomas e, às vezes, construímos modelos de caracteres que são ainda mais restritivos, por exemplo, tratando "A" e "a" como o mesmo símbolo ou tratando toda a pontuação como o mesmo símbolo. Mas, com modelos de palavras, temos pelo menos dezenas de milhares de símbolos e às vezes milhões. A grande variedade é porque não é claro o que constitui uma palavra. Em inglês, uma sequência de letras rodeada por espaços é uma palavra, mas em algumas línguas, como o chinês, as palavras não são separadas por espaços, e até mesmo em inglês

muitas decisões devem ser feitas para ter uma política clara sobre os limites da palavra: quantas palavras existem em “ne'er-do-well”? Ou em “(Tel:1-800-960-5660x123)”?

Os modelos de palavras de *n*-grama têm de lidar com palavras **fora do vobabulário**. Com modelos de caracteres, não precisamos nos preocupar com alguém inventando uma nova letra no alfabeto.[1] Mas, com modelos de palavras, há sempre a chance de uma nova palavra que não foi vista no *corpus* de treinamento, por isso precisamos modelar isso explicitamente em nosso modelo de linguagem. Isso pode ser feito acrescentando apenas uma nova palavra ao vocabulário: `<UNK>`, como palavra desconhecida. Podemos estimar o resultado de *n*-grama para `<UNK>` através deste truque: passar pelo *corpus* de treinamento e, na primeira vez que qualquer palavra individual aparecer, sendo previamente desconhecida, substituímos pelo símbolo `<UNK>`. Todos os aparecimentos subsequentes da palavra permanecem inalterados. Então, calcula-se o resultado de *n*-grama para o *corpus*, como de costume, tratando `<UNK>` como qualquer outra palavra. Assim, quando uma palavra desconhecida aparecer em um conjunto de teste, examinaremos a sua probabilidade sob `<UNK>`. Às vezes são utilizados símbolos de múltiplas palavras desconhecidas, para classes diferentes. Por exemplo, qualquer sequência de dígitos pode ser substituído por `<NUM>` ou qualquer endereço de e-mail, por `<E-MAIL>`.

Para se ter uma noção de que modelos de palavras criar, construímos modelos de unigrama, bigrama e trigrama além das palavras neste livro e, em seguida, sequências de palavras aleatoriamente amostradas dos modelos. Os resultados são

Unigrama: lógicas são como são confusão um pode direito tentar agente objetivo foi…

Bigrama: sistemas são abordagens computacionais muito semelhantes seria representado…

Trigrama: planejamento e esquedulagem estão integrados o sucesso do modelo simples de bayes é…

Mesmo com essa pequena amostra, deve ficar claro que o modelo de unigrama é uma aproximação pobre, tanto do inglês como do conteúdo de um livro de IA, e que os modelos de bigrama e trigrama são muito melhores. Os modelos estão de acordo com esta avaliação: a perplexidade era 891 para o modelo unigrama, 142 para o modelo bigrama e 91 para o modelo trigrama.

Com os princípios estabelecidos dos modelos de *n*-grama, tanto baseado em caracteres como em palavras, podemos nos voltar agora para algumas tarefas da linguagem.

22.2 CLASSIFICAÇÃO DE TEXTO

Vamos agora considerar em profundidade a tarefa de **classificação de texto**, também conhecida como **categorização**: dado algum tipo de texto, decida qual conjunto predefinido de classes pertence a ele. A identificação da linguagem e a classificação do gênero são exemplos de classificação de texto, como também é a análise do sentimento (classificação de um filme ou revisão de produto como positivo ou negativo) e a **descoberta de spam** (classificação de uma mensagem de e-mail como spam ou não spam). Uma vez que “não spam” é estranho, os pesquisadores cunharam o termo **ham** para não

spam. Podemos tratar a descoberta de spam como um problema em aprendizagem supervisionada. Um conjunto de treinamento está disponível: os exemplos positivos (spam) estão na minha pasta de spam, os exemplos negativos (ham) estão na minha caixa de entrada. Veja esta extração:

Spam: Venda por atacado de óculos modernos –57% hoje. Relógios de grife por preço baixo…
Spam: Você pode comprar ViagraFr $ 1,85 Todos os medicamentos a preços imbatíveis!…
Spam: PODEMOS TRATAR DE QUALQUER COISA QUE VOCÊ TENHA APENAS CONFIE EM NÓS …
Spam: Come.ce a ganhar* o salário que vo,cê m-erece o'btendo referên'cias apropriada,s!

Ham: O significado prático de largura da hiperárvore identificando mais…
Ham: Resumo: Vamos motivar o problema de aglomeração de identidade social:…
Ham: É bom ver você meu amigo. Ei Pedro, Foi bom saber de você. …
Ham: PDS implica convexidade do problema de otimização resultante (Kernel Ridge …

A partir dessa extração, podemos começar a ter uma ideia de quais seriam as boas características para incluir no modelo de aprendizagem supervisionada. *n*-gramas de palavras e como "preço baixo" e "Você pode comprar" parecem indicadores de spam (embora tenham probabilidade diferente de zero de ser ham também). Atributos de caracteres também parecem importantes: é mais provável que o spam esteja todo em letras maiúsculas e tenha pontuações incorporadas às palavras. Aparentemente, os spammers pensaram que a palavra bigrama "você merece" seria muito indicativa de spam e então escreveram "vo,cê m-erece". Um modelo de caractere deveria descobrir isso. Ou poderíamos criar um modelo de *n*-grama de caracteres completo de spam e ham ou poderíamos criar as características manualmente como "número de sinais de pontuação incorporados às palavras".

Observe que temos duas formas complementares de falar sobre classificação. Na abordagem de modelagem de linguagem, definimos um modelo de linguagem de *n*-grama para $\mathbf{P}(Mensagem \mid spam)$ pelo treinamento em uma a pasta de spam, e um modelo $\mathbf{P}(Message \mid ham)$ pelo treinamento em uma caixa de entrada. Então, podemos classificar uma nova mensagem com a aplicação da regra de Bayes:

$$\underset{c \in \{spam, ham\}}{\text{argmax}} \; P(c \mid mensagem) = \underset{c \in \{spam, ham\}}{\text{argmax}} \; P(mensagem \mid c)\, P(c).$$

onde $P(c)$ é estimado apenas pela contagem do número total de mensagens de spam e ham. Essa abordagem funciona bem para a descoberta de spam, tal como aconteceu para a identificação do idioma.

Na abordagem de aprendizagem de máquina representamos a mensagem como um conjunto de pares de característica/valor e aplicamos um algoritmo de classificação h à característica de vetor $\mathbf{X}$. Podemos tornar as abordagens de modelagem de linguagem e de aprendizagem de máquina compatíveis pensando em *n*-gramas como características. É mais fácil visualizar com um modelo unigrama. As características são as palavras no vocabulário: "a," "aardvark"…, e os valores são o número de vezes que cada palavra aparece na mensagem. Isso torna o vetor de característica grande e esparso. Se existirem 100 mil palavras no modelo de linguagem, o vetor de características terá comprimento 100.000, mas para uma mensagem de e-mail curta quase todas as características terão

resultado zero. Essa representação do unigrama tem sido chamada de modelo de **saco de palavras**. Você pode pensar sobre o modelo como colocar as palavras de treinamento do *corpus* em um saco e em seguida selecionar as palavras uma de cada vez. A noção de ordem das palavras é perdida; um modelo de unigrama oferece a mesma probabilidade para qualquer permutação de um texto. Modelos de *n*-grama de ordem superior mantêm alguma noção local da ordem das palavras.

Com bigramas e trigramas, o número de características é elevado ao quadrado ou ao cubo e podemos acrescentar outras características não *n*-grama: o tempo em que a mensagem foi enviada, se uma URL ou uma imagem faz parte da mensagem, um número de identificação do remetente da mensagem, o número do remetente das mensagens ham e spam anteriores, e assim por diante. A escolha das características é a parte mais importante na criação de um bom detector de spam — mais importante do que a escolha do algoritmo para processamento das características. Em parte, isso é porque existe grande quantidade de dados de treinamento; por isso, se pudermos propor uma característica, os dados poderão determinar com precisão se é boa ou não. É necessário atualizar constantemente as características de atualização porque a detecção de spam é uma **tarefa adversarial**; os spammers modificam seus spams em resposta às alterações no detector de spam.

Pode ser caro executar algoritmos em um vetor de características muito grandes, por isso muitas vezes um processo de **seleção de características** é usado para manter apenas as características que melhor distinguem entre spam e ham. Por exemplo, o bigrama “of the” é frequente em inglês e pode ser igualmente frequente em spam e ham, então não há sentido em contá-lo. Muitas vezes, os cem melhores ou mais característicos fazem um bom trabalho de distinção entre classes.

Uma vez escolhido um conjunto de características, podemos aplicar qualquer uma das técnicas de aprendizagem supervisionada que vimos; as mais populares para a categorização de texto incluem *k*-vizinhos mais próximos, máquinas de vetor de suporte, árvores de decisão, Bayes ingênuo e regressão logística. Todos eles têm sido aplicados para descoberta de spam, geralmente com precisão na faixa de 98 -99%. Com um conjunto de recursos projetado cuidadosamente, a precisão pode ultrapassar 99,9%.

22.2.1 Classificação por compressão de dados

Outra maneira de pensar sobre a classificação é como um problema na **compressão de dados**. Um algoritmo de compressão sem perdas considera uma sequência de símbolos, detecta padrões repetidos no mesmo e escreve uma descrição da sequência mais compacta do que a original. Por exemplo, o texto “0,142857142857142857” pode ser comprimido para “0, [142857]*3”. Os algoritmos de compressão trabalham com a construção de dicionários de subsequências do texto e depois se referem à entradas no dicionário. O exemplo aqui só tinha uma entrada no dicionário, “142857”.

Com efeito, os algoritmos de compressão estão criando um modelo de linguagem. O algoritmo LZW em particular modela diretamente uma distribuição de probabilidade de entropia máxima. Para fazer a classificação por compressão, primeiro reunimos todas as mensagens de spam de treinamento e as comprimimos como uma unidade. Fazemos o mesmo para o ham. Então, quando uma nova mensagem é fornecida para classificar, nós a anexamos às mensagens de spam e comprimimos o

resultado. Também a anexamos ao ham e comprimimos. Qualquer classe que comprima melhor — adiciona o menor número de bytes adicionais para a nova mensagem — é a classe prevista. A ideia é que uma mensagem de spam tenderá a compartilhar entradas de dicionário com outras mensagens de spam e, portanto, vai comprimir melhor quando anexada a uma coleção que já contém o dicionário de spam.

Experimentos com classificação baseada em compressão em alguns dos *corpora*-padrão para classificação do texto — 20 conjuntos de dados de novos grupos *corpora* Reuters-10, *corpora* do setor da indústria — indicam que, enquanto a execução da compressão de algoritmos prontos para uso como gzip, RAR e LZW pode ser bastante lenta, sua precisão é comparável à de algoritmos de classificação tradicional. Isso é interessante em seu próprio direito e também serve para destacar que há promessa para os algoritmos que usam caractere *n*-gramas diretamente sem pré-processamento de texto ou seleção de característica: eles parecem estar capturando alguns padrões reais.

22.3 RECUPERAÇÃO DE INFORMAÇÃO

A **recuperação de informação** é a tarefa de encontrar documentos que são relevantes para a necessidade de um usuário de obter informação. Os exemplos mais conhecidos de sistemas de recuperação de informação são os mecanismos de busca na World Wide Web. Um usuário da Web pode digitar uma consulta, como [livro IA][2] em um mecanismo de busca e obter uma lista de páginas relevantes. Nesta seção, veremos como esses sistemas são construídos. Um sistema de recuperação de informação (doravante **RI**) pode ser caracterizada por

1. Um ***corpus* de documentos**. Cada sistema deve decidir o que quer tratar como documento: um parágrafo, uma página ou um texto de várias páginas.
2. **Consultas colocadas em linguagem de consulta**. Uma consulta especifica sobre o que o usuário quer saber. A linguagem de consulta pode ser apenas uma lista de palavras, tais como [livro IA] ou pode especificar uma sintagma com palavras que devem ser adjacentes, como em [“livro IA”] e conter operadores booleanos como em [IA E livro], incluir operadores não booleanos, tais como [IA PERTO livro] ou [livro de IA site: www.aaai.org].
3. **Um conjunto de resultados**. Esse é o subconjunto de documentos que o sistema de RI julga ser **relevante** para a consulta. Por *relevante* queremos dizer provável que seja de utilidade para a pessoa que fez a consulta, para a necessidade de informação específica expressa na consulta.
4. **Apresentação do conjunto de resultados**. Isso pode ser tão simples como uma lista ordenada de títulos de documentos ou tão complexo como um mapa de cores de rotação do conjunto de resultados projetado em um espaço tridimensional, processado como uma exibição bidimensional.

Os primeiros sistemas de RI trabalhavam com um **modelo de palavra-chave booleano**. Cada palavra na coleção do documento era tratada como uma característica booleana que era verdadeira em um documento se a palavra ocorresse no documento e falsa se não ocorresse. Assim, a característica “recuperação” seria verdadeira para o capítulo atual, mas falsa para o Capítulo 15. A

linguagem de consulta era a de expressões booleanas sobre as características. Um documento só será relevante se a expressão for avaliada como verdadeira. Por exemplo, a consulta [informação E recuperação] será verdadeira para o capítulo atual e falsa para o Capítulo 15.

Esse modelo tem a vantagem de ser simples de explicar e implementar. No entanto, tem algumas desvantagens. Primeiro, o grau de relevância de um documento é um único bit, de forma que não há nenhuma orientação sobre como ordenar os documentos relevantes para a apresentação. Em segundo lugar, expressões booleanas não são familiares a usuários que não são programadores ou lógicos. Os usuários não acham intuitivo que, quando querem saber sobre a agricultura nos estados do Kansas e Nebraska, precisem emitir uma consulta [agricultura (Kansas OU Nebraska)]. Terceiro, pode ser difícil formular uma consulta apropriada, mesmo para um usuário experiente. Suponha que tentemos [informação E recuperação E modelos E otimização] e obtemos um conjunto de resultados vazio. Poderíamos tentar [informação OU recuperação OU modelos OU otimização], mas, se retornar muitos resultados, é difícil saber o que tentar depois.

22.3.1 Funções de pontuação para RI

A maioria dos sistemas de RI abandonou o modelo booleano e começou a usar modelos baseados em estatísticas de contagem de palavras. Descreveremos a **função de pontuação BM25**, que vem do projeto Okapi de Stephen Robertson e Karen Sparck Jones do London's City College, e tem sido usada em mecanismos de busca, tal como o projeto Lucene de código livre.

A função de pontuação retorna uma pontuação numérica a partir de um documento e de uma consulta; os documentos mais relevantes têm as maiores pontuações. Na função BM25, a pontuação é um combinação linear ponderada das pontuações para cada uma das palavras que compõem a consulta. Três fatores afetam o peso de um termo da consulta: primeiro, a frequência com que um termo de consulta aparece em um documento (também conhecido como *FT* para frequência do termo). Para a consulta [agricultura em Kansas], os documentos que mencionam "agricultura" frequentemente terão maior pontuação. Segundo, a frequência inversa do documento com o termo ou *FID*. A palavra "em" aparece em quase todos os documentos, por isso tem frequência alta em documentos e, assim, um documento com frequência baixa, portanto, não é tão importante para a consulta como "agricultura" ou "Kansas". Terceiro, o comprimento do documento. Um documento de um milhão de palavras provavelmente vai mencionar todas as palavras da consulta, mas pode realmente não ser sobre a consulta. Um documento breve que menciona todas as palavras seria um candidato muito melhor.

A função BM25 considera todos esses três itens. Vamos assumir que tenhamos criado um índice de N documentos no *corpus* para que possamos pesquisar $TF(q_i, d_j)$, a quantidade do número de vezes que a palavra q_i aparece no documento d_j. Assumimos também uma tabela de contagem de frequência de documento, $DF(q_i)$, que fornece o número de documentos que contém a palavra q_i. Então, dado um documento d_j e uma consulta que consiste nas palavras $\theta_{1:n}$, temos

$$BM25(d_j, q_{1:N}) = \sum_{i=1}^{N} IDF(q_i) \cdot \frac{TF(q_i, d_j) \cdot (k+1)}{TF(q_i, d_j) + k \cdot (1 - b + b \cdot \frac{|d_j|}{L})},$$

onde $|d_j|$ é o comprimento do documento d_j em palavras e L é o comprimento médio do documento no *corpus*: $L = \sum_i |d_i|/N$.Temos dois parâmetros, k e b, que podem ser relacionados por validação cruzada; os valores típicos são $k = 2{,}0$ e $b = 0{,}75$. O IDF(q_i) é o documento de frequência inversa da palavra q_i, dado por

$$IDF(q_i) = \log \frac{N - DF(q_i) + 0{,}5}{DF(q_i) + 0{,}5}.$$

Certamente seria impraticável aplicar a função de pontuação BM25 para todo o documento no *corpus*. Em vez disso, os sistemas criam um **índice** antecipadamente que lista, para cada palavra do vocabulário, os documentos que contêm a palavra. Isso é chamado de **lista de hits** da palavra. Então, dada uma consulta, cruzam-se as listas de hits das palavras da consulta e só se pontuam os documentos encontrados no cruzamento.

22.3.2 Avaliação do sistema de RI

Como sabemos se um sistema de RI está tendo bom desempenho? Pode-se tentar um experimento no qual se fornece ao sistema um conjunto de consultas, e os resultados dos conjuntos são pontuados de acordo com o julgamento de relevância dos seres humanos. Tradicionalmente, havia duas medidas utilizadas na pontuação: cobertura e precisão. Vamos explicá-las com a ajuda de um exemplo. Imagine que um sistema RI retornou um conjunto de resultados para uma única consulta, para a qual sabemos quais documentos são relevantes e quais não são, de um *corpus* com 100 documentos. A quantidade de documentos em cada categoria é dada na tabela a seguir:

	No conjunto de resultados	Fora do conjunto de resultados
Relevantes	30	20
Não relevantes	10	40

A **precisão** mede a proporção de documentos no conjunto de resultados que são realmente relevantes. No nosso exemplo, a precisão é de 30/(30 + 10) = 0,75. A taxa de falsos positivos é de 1 − 0,75 = 0, 25. A **cobertura** mede a proporção de todos os documentos relevantes na coleção que estão no conjunto de resultados. No nosso exemplo, a cobertura é 30/(30 + 20) = 0,60. A taxa de falso negativo é de 1 − 0,60 = 0,40. Em uma coleção de documentos muito grande, como na World Wide Web, a cobertura é difícil de calcular porque não há nenhuma maneira fácil de examinar cada página da Web para relevância. Tudo o que podemos fazer é estimar a cobertura por amostragem ou ignorá-la completamente e apenas julgar a precisão. No caso de um mecanismo de busca da Web, pode haver milhares de documentos no conjunto de resultados; assim, faz mais sentido medir a precisão de vários tamanhos diferentes, tal como “P@10” (precisão entre os 10 resultados

principais) ou “P@50”, em vez de estimar a precisão em todo o conjunto de resultados.

É possível compensar a precisão em relação à cobertura variando o tamanho do conjunto de resultados retornado. Em casos extremos, um sistema que retorna todos os documentos na coleção de documentos é a garantia de uma cobertura de 100%, mas terá baixa precisão. Já um sistema pode retornar um único documento e ter cobertura baixa, mas uma chance decente de precisão de 100%. Um resumo de ambas as medidas é a pontuação de F_1, um número único que é a média harmônica de precisão e cobertura, $2PR/(P + R)$.

22.3.3 Refinamentos de RI

Há muitos refinamentos possíveis para o sistema descrito aqui, e de fato os mecanismos de pesquisa da Web estão continuamente atualizando seus algoritmos à medida que descobrem novas abordagens e à medida que a Web cresce e muda.

Um refinamento comum é um modelo melhor do efeito do tamanho do documento sobre a relevância. Singhal *et al*. (1996) observaram que esquemas de normalização de tamanho de documentos simples tendem a favorecer os documentos curtos demais e não favorecem o suficiente os longos. Eles propõem um esquema de normalização do comprimento do documento *pivô;* a ideia é que o pivô seja o comprimento do documento em que a normalização de estilo antigo está correta; documentos mais curtos recebem um incentivo, e os mais longos obtêm uma penalidade.

A função de pontuação BM25 utiliza um modelo de palavra que trata todas as palavras como completamente independentes, mas sabemos que algumas palavras são correlatas: “divã” está intimamente relacionado com “divã” e “sofá”. Muitos sistemas de RI tentam explicar essas correlações.

Por exemplo, se a consulta for [sofá], será uma pena excluir do conjunto de resultados aqueles documentos que mencionam “SOFÁ” ou “sofás” mas não “sofá”. A maioria dos sistemas de RI não é **sensível à caixa** alta e baixa no caso de “SOFÁ” ou “sofá”, e alguns usam algoritmos de **radicalização** para reduzir “sofás” para a forma da raiz “sofá”, tanto em consulta como em documentos. Isso produz normalmente um pequeno aumento na cobertura (da ordem de 2% para o inglês). No entanto, pode prejudicar a precisão. Por exemplo, transformar para o radical em inglês “stocking” para “stock” tenderá a diminuir a precisão para consultas sobre meias ou instrumentos financeiros, embora pudesse melhorar a cobertura para consultas sobre armazenagem. Algoritmos de radicalização baseados em regras (por exemplo, remover “-ing”) não podem evitar esse problema, mas algoritmos baseados em dicionários (não retire o “-ing” se a palavra já está listada no dicionário) podem. Enquanto reduzir à raiz da palavra tem efeito pequeno em inglês, é mais importante em outras línguas. Em alemão, por exemplo, não é incomum ver palavras como “Lebensversicherungsgesellschaftsangestellter” (funcionário da empresa de seguros de vida). Idiomas como o finlandês, o turco, o inuíte e o *yupik* têm regras morfológicas recursivas que, em princípio, geram palavras de comprimento ilimitado.

O próximo passo é reconhecer **sinônimos**, como “divã”, para “sofá”. Tal como para reduzir ao mesmo radical, tem o potencial de ganhos pequenos em cobertura, mas pode interferir na precisão. Um usuário que fornece a consulta [Tim Couch] quer ver resultados sobre o jogador de futebol e não

sobre sofás. O problema é que “os idiomas abominam sinônimos absolutos, assim como a natureza abomina o vácuo” (Cruse, 1986). Isto é, sempre que existirem duas palavras que significam a mesma coisa, usuários da língua conspirarão para evoluir os significados para remover a confusão. Palavras relacionadas que não são sinônimos também desempenham papel importante na classificação — termos como “couro”, “madeira” ou “moderno” podem servir para confirmar que o documento realmente é sobre “sofá”. Sinônimos e palavras relacionadas podem ser encontrados em dicionários ou procurando por correlações em documentos ou consultas — se acharmos que muitos usuários que pedem pela consulta [divã novo] dão prosseguimento com a consulta [sofá novo], podemos alterar no futuro [divã novo] para ser [sofá novo OU divã novo].

Como refinamento final, a RI pode ser melhorada considerando os **metadados** — dados externos ao texto do documento. Os exemplos incluem palavras-chave fornecidas por seres humanos e dados de publicação. Na Web, os **links** de hipertexto entre documentos são fonte crucial de informação.

22.3.4 Algoritmo PageRank

O **PageRank**[3] foi uma de duas ideias originais que diferenciaram a busca do Google à parte de outros mecanismos de busca da Web quando foi introduzida em 1997. (A outra inovação foi o uso de texto âncora — o texto sublinhado em um hiperlink — para indexar uma página, mesmo que o texto âncora esteja em uma página *diferente* da que está sendo indexada.) O PageRank foi inventado para resolver o problema da tirania da pontuação de *FT*: se a consulta for [IBM], como podemos ter certeza de que a homepage da IBM, ibm.com, é o primeiro resultado, mesmo se outra página menciona o termo “IBM” com mais frequência? A ideia é que ibm.com tem muitas ligações de entrada (links para a página), por isso deve ter uma classificação mais alta: cada link de entrada é um voto para a qualidade da página vinculada. Mas, se contarmos apenas os links de entrada, será possível para um spammer da Web criar uma rede de páginas e tê-las todas apontando para uma página de sua escolha, aumentando a pontuação dessa página. Portanto, o algoritmo PageRank foi projetado para ponderar links de sites de alta qualidade mais pesadamente. O que é um site de alta qualidade? Aquele que está ligado a outros sites de alta qualidade. A definição é recursiva, mas veremos que a recursão estabiliza de forma adequada. O PageRank para uma página *p* é definido como:

$$PR(p) = \frac{1-d}{N} + d\sum_i \frac{PR(in_i)}{C(in_i)},$$

onde $PR(p)$ é o PageRank da página *p*, *N* é o número total de páginas no corpus, *ini* são as páginas que se ligam a *p* e $C(ini)$ é a contagem do número total de links de saída na página in_i. A constante *d* é um fator de amortecimento. Ela pode ser entendida através do **modelo de navegação aleatório**: imagine um surfista da Web que inicia em alguma página aleatória e começa a navegar. Com probabilidade *d* (vamos supor que $d = 0{,}85$) o surfista clica em um dos links da página (escolhendo uniformemente entre eles) e com probabilidade $1 - d$ fica entediado com a página e reinicia em uma página aleatória em qualquer lugar na Web. O PageRank da página *p* é então a probabilidade de que o navegador aleatório estará na página *p* em qualquer ponto no tempo. O PageRank pode ser

calculado por um procedimento iterativo: iniciar com todas as páginas com $PR(p) = 1$ e iterar o algoritmo, atualizando as ordens até que convirjam.

22.3.5 O algoritmo HITS

O algoritmo Hyperlink-Induced Topic Search, também conhecido como “hubs e autoridades” ou HITS, é outro algoritmo influente na análise de links (veja a Figura 22.1). O HITS difere do PageRank de várias maneiras. Primeiro, é uma medida dependente de consulta: classifica páginas de acordo com uma consulta. Isso significa que deve ser calculado de novo para cada consulta — um fardo computacional que a maioria dos mecanismos de busca tem optado por não assumir. Dada uma consulta, o HITS primeiro encontra um conjunto de páginas que são relevantes para a consulta, pela interseção de listas de hits de palavras de consulta e, em seguida, adiciona as páginas nos links de vizinhança dessas páginas — páginas que apontam para/são apontadas de uma das páginas do conjunto original relevante.

```
função HITS (consulta) retorna páginas com hub e números de autoridade
   páginas ← EXPANDIR-PÁGINAS (PÁGINAS-RELEVANTES (consulta))
   para cada p em páginas faça
      p.AUTORIDADE ← 1
      p.HUB ←1
   repetir até a convergência faça
      para cada p em páginas faça
         p.AUTORIDADE ← ∑i INLINKi (p).HUB
         p.HUB ← ∑i OUTLINKi (p).AUTHORITY
      NORMALIZAR (páginas)
   retornar páginas
```

Figura 22.1 O algoritmo HITS calcula hubs e autoridades com respeito a uma consulta. PÁGINAS-RELEVANTES busca as páginas que correspondem à consulta e EXPANDIR-PÁGINAS acrescenta cada página que liga a uma das páginas relevantes ou está ligada. NORMALIZAR divide a pontuação de cada página pela soma dos quadrados da pontuação de todas as páginas (separadamente, tanto para a pontuação de hubs como de autoridade).

Cada página desse conjunto é considerada uma **autoridade** sobre a consulta na medida em que outras páginas do conjunto relevante apontem para ela. A página é considerada um **hub** na medida em que aponta para outras páginas de autoridade no conjunto relevante. Assim como com o PageRank, não queremos apenas contar o número de links; queremos dar mais valor aos hubs e autoridades de alta qualidade. Assim, como com o PageRank, iteramos um processo que atualiza a pontuação de autoridade de uma página para ser a soma das pontuações do hub das páginas que apontam para ela, e a pontuação do hub para ser a soma das pontuações de autoridade das páginas para as quais ele aponta. Se normalizarmos as pontuações e repetirmos *k* vezes, o processo convergirá.

O PageRank e o HITS desempenharam papéis importantes no desenvolvimento de nossa compreensão da recuperação de informação da Web. Esses algoritmos e suas extensões são usados em bilhões de classificação de consultas diariamente à medida que os mecanismos de busca constantemente desenvolvem melhores formas de extrair sinais ainda mais finos de relevância da busca.

22.3.6 Respostas a perguntas

A recuperação de informação é a tarefa de encontrar documentos que são relevantes para uma consulta, podendo a consulta ser uma pergunta ou apenas um tema de uma área ou um conceito. **Respostas a perguntas** é uma técnica um pouco diferente, na qual a consulta é realmente uma pergunta, e a resposta não é uma lista ordenada de documentos, mas uma resposta curta — uma sentença ou mesmo apenas um sintagma. Existem sistemas de perguntas e respostas baseadas em PLN (processamento de linguagem natural) desde 1960, mas apenas a partir de 2001 esses sistemas usaram recuperação de informação da Web para aumentar radicalmente a sua amplitude de cobertura.

O sistema ASKMSR (Banko *et al*., 2002) é um sistema típico de pergunta e resposta baseado na Web. É baseado na intuição de que a maioria das perguntas será respondida muitas vezes na Web; assim, perguntas e respostas devem ser consideradas como um problema de precisão, não de cobertura. Não precisamos lidar com todas as maneiras diferentes como uma resposta pode ser formulada; temos apenas de encontrar uma delas. Por exemplo, considere a consulta [Quem matou Abraham Lincoln?]. Suponha que um sistema teve de responder a essa pergunta com acesso apenas a uma enciclopédia única, cuja entrada para Lincoln informava

> John Wilkes Booth alterou a história com uma bala. Ele será lembrado para sempre como o homem que acabou com a vida de Abraham Lincoln.

Para utilizar essa passagem para responder à pergunta, o sistema teria de saber que acabar com uma vida pode ser um assassinato, que "ele" refere-se a Booth, e vários outros fatos linguísticos e semânticos.

O ASKMSR não tenta esse tipo de sofisticação — não sabe nada sobre referência de pronome ou sobre matar, ou qualquer outro verbo. Ele sabe 15 tipos diferentes de perguntas e como elas podem ser reescritas como consultas para um mecanismo de busca. Ele sabe que [Quem matou Abraham Lincoln] pode ser reescrito como a consulta [* matou Abraham Lincoln] e como [Abraham Lincoln foi morto por *]. Ele emite essas consultas reescritas e examina os resultados que retornam — não as páginas inteiras da Web, apenas os resumos curtos de texto que aparecem próximos dos termos da consulta. Os resultados são divididos em 1-, 2- e 3-gramas e controlados por frequência nos conjuntos de resultados e de pesos: um n-grama que voltou reescrito de uma consulta muito específica (tal como a consulta de correspondência exata do sintagma ["Abraham Lincoln foi morto por *"]) ganharia mais peso do que uma a partir de uma consulta geral reescrita como [Abraham OU Lincoln OU morto]. Seria de esperar que "John Wilkes Booth" estivesse altamente classificado entre as n-gramas recuperadas, mas também estaria "Abraham Lincoln" e "o assassinato de" e "Teatro Ford".

Uma vez que *n*-gramas forem pontuados, são filtrados pelo tipo esperado. Se a consulta original começa por "quem", filtramos nomes de pessoas, por "quantos" filtramos números, "quando", uma data ou hora. Há também um filtro que diz que a resposta não deve ser parte da questão; juntos, devem permitir-nos retornar "John Wilkes Booth" (e não "Abraham Lincoln") como a resposta de pontuação mais alta.

Em alguns casos, a resposta será mais longa que três palavras; já que as respostas componentes só vão até 3-gramas, uma resposta longa teria de ser montada a partir de pedaços mais curtos. Por exemplo, em um sistema que use apenas bigramas, a resposta "John Wilkes Booth" poderia ser montada a partir de pedaços com pontuação alta "John Wilkes" e "Wilkes Booth".

Na Text Retrieval Evaluation Conference (TREC), o ASKMSR foi classificado como um dos melhores sistemas, batendo concorrentes com a capacidade de fazer a compreensão muito mais complexa de linguagem. O ASKMSR depende da amplitude do conteúdo na Web, em vez de sua própria profundidade de compreensão. Não será capaz de lidar com padrões de inferência complexos como associar "quem matou" com "terminou a vida de". Mas sabe que a Web é tão extensa que pode se dar ao luxo de ignorar passagens como essa e esperar por uma simples passagem que possa manipular.

22.4 EXTRAÇÃO DE INFORMAÇÃO

A **extração de informação** é o processo de aquisição de conhecimento passando os olhos em um texto e procurando por ocorrências de uma classe particular de objetos e de relações entre objetos. A tarefa típica é extrair exemplos de endereços de páginas da Web, com campos de banco de dados de rua, cidade, estado e código postal; ou casos de tempestades a partir de informações meteorológicas, com campos para temperatura, velocidade do vento e precipitação. Em um domínio limitado, isso pode ser feito com alta precisão. Como o domínio fica mais geral, são necessários modelos linguísticos e técnicas de aprendizagem mais complexas. Veremos no Capítulo 23 como definir modelos de linguagem complexos a partir da estrutura de sintagmas (sintagmas nominais e verbais) em inglês. Mas até aqui não vimos modelos completos desse tipo; portanto, para as necessidades limitadas de extração de informação, definimos modelos limitados que se aproximam do modelo completo em inglês e se concentram apenas nas partes necessárias para a tarefa à mão. Os modelos que descrevemos nesta seção são aproximações da mesma forma que o modelo lógico único1-CNF na Figura 7.21 é uma aproximação do modelo lógico completo e sinuoso.

Nesta seção, descrevemos seis abordagens diferentes para a extração de informações, a fim de aumentar a complexidade em várias dimensões: de determinística a estocástica, de domínio específico a geral, de artesanal a aprendido e de pequena escala a grande escala.

22.4.1 Autômatos de estado finito para a extração de informações

O tipo mais simples de sistema de extração de informação é o sistema de **extração baseada em atributos**, que assume que todo texto se refere a um único objeto e a tarefa é extrair atributos desse

objeto. Por exemplo, mencionamos na Seção 12.7 o problema da extração do texto “IBM ThinkBook 970. Nosso preço: $399,00” como o conjunto de atributos {Fabricante = IBM, Modelo = ThinkBook970 Preço = $ 399,00}. Podemos resolver esse problema através da definição de um modelo (também conhecido como padrão) para cada atributo que gostaríamos de extrair. O modelo é definido por um autômato finito cujo exemplo mais simples é a **expressão regular**, ou regex. Expressões regulares são usadas em comandos Unix, como grep, em linguagens de programação como Perl e em processadores de texto como o Microsoft Word. Os detalhes variam ligeiramente de uma ferramenta para outra e por isso são mais bem aprendidos através de um manual apropriado, mas aqui vamos mostrar como construir um modelo de expressão regular para preços em dólares:

`[0-9]`	corresponde a qualquer dígito entre 0 e 9
`[0-9]+`	corresponde a um ou mais dígitos
`[.] [0-9] [0-9]`	corresponde a um ponto seguido de dois dígitos
`([.][0-9] [0-9])?`	corresponde a um ponto seguido de dois dígitos ou nada
`[$] [0-9]+([.][0-9] [0-9])?`	corresponde a $249,99 ou $1,23 ou $1.000.000 ou...

Os modelos são geralmente definidos em três partes: um prefixo regex, um objetivo regex e um sufixo regex. Para os preços, o objetivo regex é como anteriormente, o prefixo procura por sequências como “preço:” e o sufixo pode estar vazio. A ideia é que algumas pistas sobre um atributo vem do valor do atributo em si, e alguns vêm a partir do texto circundante.

Se uma expressão regular para um atributo corresponder ao texto exatamente uma vez, podemos tirar a parte do texto que é o valor do atributo. Se não houver correspondência, tudo o que podemos fazer é dar um valor-padrão ou deixar o atributo em falta, mas se houver várias correspondências precisamos de um processo para escolher entre elas. Uma estratégia é ter vários modelos para cada atributo, ordenados por prioridade. Assim, por exemplo, o modelo cuja primeira prioridade é o preço pode procurar pelo prefixo “nosso preço:”; se não for encontrado, procuramos pelo prefixo “preço:” e, se não for encontrado, pelo prefixo vazio. Outra estratégia é pegar todas as correspondências e encontrar alguma maneira de escolher entre elas. Por exemplo, poderíamos pegar o menor preço que está dentro de 50% do preço mais alto. Será selecionado $78,00 como objetivo do texto “Lista de preço $99,00, preços de venda especiais $78,00, transporte $3,00”.

Uma etapa acima dos sistemas de extração baseados em atributos estão os sistemas de **extração relacional**, que lidam com vários objetos e com as relações entre eles. Assim, quando esses sistemas veem o texto “$249,99”, eles precisam determinar não apenas que é um preço, mas também que objeto tem esse preço. Um sistema típico baseado em extração relacional é o FASTUS, que lida com histórias de notícias sobre fusões e aquisições corporativas. Pode ser lida a história

> A Bridgestone Sports Co. informou que sexta-feira estabeleceu uma *joint venture* em Taiwan com a firma local e uma casa de comércio japonesa para produzir tacos de golfe para serem enviados ao Japão.

e extrair as relações:

$$e \in \mathit{JointVentures} \wedge \mathit{Produto}\ (e, \text{“clubes de golfe”}) \wedge \mathit{Data}\ (e, \text{“sexta-feira”})$$
$$\wedge\ \mathit{Membro}\ (e, \text{“Bridgestone Sports Co”}) \wedge \mathit{Membros}\ (e, \text{“firma local”})$$
$$\wedge\ \mathit{Membro}\ (e, \text{“uma casa de comércio japonesa”}).$$

Um sistema de extração relacional pode ser construído como uma série de **transdutores de estado finito em cascata**. Ou seja, o sistema consiste em uma série de pequenos autômatos de estados finitos eficientes (FSAs), em que cada autômato recebe um texto como entrada, transduz o texto em um formato diferente e o passa adiante ao autômato seguinte. O FASTUS consiste em cinco etapas:

1. Tokenização
2. Manuseio de palavras complexas
3. Manuseio de grupos básicos
4. Manuseio de sintagmas complexos
5. Fusão de estruturas.

A primeira etapa do FASTUS é a tokenização, que segmenta o fluxo de caracteres em tokens (palavras, números e pontuação). Para o inglês, a tokenização pode ser bastante simples; apenas a separação de caracteres em espaços em branco ou pontuação é um trabalho bastante bom. Alguns tokenizadores também lidam com linguagens de marcação, como HTML, SGML e XML.

A segunda fase lida com **palavras complexas**, incluindo colocações como "estabeleceu" e "*joint venture*", bem como nomes próprios como "Bridgestone Sports Co.". São reconhecidos por uma combinação de entradas lexicais e regras de gramática de estado finito. Por exemplo, o nome da empresa pode ser reconhecido pela regra

PalavraMaiúscula+ ("Company" | "Co" | "Inc" | "Ltd").

A terceira etapa trata de **grupos básicos**, ou seja, grupos nominais e verbais. A ideia é fatiá-los em unidades que serão geridas pelas fases posteriores. Veremos como escrever uma descrição complexa de frases nominais e verbais no Capítulo 23, mas aqui temos regras simples que apenas se aproximam da complexidade do inglês, porém com a vantagem de ser representáveis por autômatos de estado finito. Um exemplo de sentença iria emergir dessa fase como a seguinte sequência de grupos rotulados:

```
1 NG: Bridgestone Sports Co.     10 NG: uma firma local
2 VG: informou                   11 CJ: e
3 NG: sexta-feira                12 NG: uma casa de comércio japonesa
4 NG: que                        13 VG: para produzir
5 VG: estabeleceu                14 NG: tacos de golfe
6 NG: uma joint venture          15 VG: para ser enviado
7 PR: em                         16 PR: ao
8 NG: Taiwan                     17 NG: Japão
9 PR: com
```

NG aqui significa grupo nominal;VG, grupo verbal; PR é preposição e CJ é conjunção.

A quarta etapa combina os grupos básicos em **frases complexas**. Mais uma vez, o objetivo é ter regras que são estados finitos e, portanto, podem ser processadas rapidamente e resultam em frases de saída sem ambiguidade (ou quase sem ambiguidade). Um tipo de regra de combinação trata eventos de domínio específico. Por exemplo, a regra

Empresa+ Estabeleceu *JointVenture* (“com” Empresa+)?

captura uma maneira de descrever a formação de uma *joint venture*. Esse estágio é o primeiro na cascata onde a saída é colocada em um modelo de banco de dados, bem como no fluxo de saída. A fase final **funde estruturas** que foram construídas na etapa anterior. Se a frase seguinte diz “A *joint venture* vai iniciar a produção em janeiro”, nota-se nessa etapa que há duas referências a uma *joint venture* e que deve ser fundida em uma só. Esse é um exemplo de **problema da incerteza de identidade** discutido na Seção 14.6.3.

Em geral, a extração de informações com base em modelo de estado finito funciona bem para um domínio restrito em que é possível predeterminar que assuntos serão discutidos e como eles serão mencionados. O modelo de transdutor em cascata ajuda a modularizar o conhecimento necessário, facilitando a construção do sistema. Esses sistemas funcionam muito bem quando estão em um texto de engenharia reversa que foi gerado por um programa. Por exemplo, um site de compras na Web é gerado por um programa que tira entradas de banco de dados e os formata em páginas da Web; um extrator baseado em modelo, em seguida, recupera o banco de dados original. A extração de informações de estado finito não é tão bem-sucedida na recuperação de informação em formatos altamente variáveis, tal como um texto escrito por seres humanos sobre uma variedade de assuntos.

22.4.2 Modelos probabilísticos para a extração de informação

Quando se deve experimentar a extração de informações a partir de entrada ruidosa ou variada, a simples abordagem do estado finito é fraca. É muito difícil conseguir todas as regras e suas prioridades corretamente; é melhor utilizar um modelo probabilístico em vez de um modelo baseado em regras. O modelo probabilístico mais simples para sequências com o estado oculto é o modelo oculto de Markov, ou MOM.

Lembre-se da Seção 15.3 que um MOM modela uma progressão através de uma sequência de estados ocultos, $\mathbf{x}_t$, com uma observação $\mathbf{e}_t$ em cada etapa. Para aplicar MOMs para extração de informações, podemos construir um grande MOM para todos os atributos ou construir um MOM separado para cada atributo. Faremos o segundo. As observações são as palavras do texto, e os estados ocultos indicam se estamos na parte do alvo, prefixo ou sufixo do modelo de atributo, ou em segundo plano (não faz parte do modelo). Por exemplo, aqui está um texto breve e o caminho mais provável (Viterbi) para o texto de dois MOMs, um treinado para reconhecer o locutor de um comunicado e outro treinado para reconhecer datas. O “-“ indica um estado em segundo plano:

Texto:	Haverá	um	seminário	pelo	Dr.	Andrew	McCallum	na	sexta-feira
Locutor:	-	-	PRÉ	PRÉ	DESTINO	DESTINO	DESTINO	PÓS	-
Data: -	-	-	-	-	-	-	-	PRÉ	DESTINO

Para a extração, os MOMs têm duas grandes vantagens sobre FSAs. Primeiro, os MOMs são probabilísticos e, portanto, tolerantes ao ruído. Em uma expressão regular, se um único caractere esperado estiver ausente, o regex falha; com os MOMs há uma degradação elegante com relação a caracteres/palavras ausentes e temos uma probabilidade indicando o grau de correspondência, não apenas uma falha/correspondência booleana. Em segundo lugar, os MOMs podem ser treinados a

partir dos dados, pois não necessitam de modelos de engenharia trabalhosos e, assim, podem ser mantidos atualizados mais facilmente à medida que o texto se altera ao longo do tempo.

Observe que assumimos certo nível de estrutura em nossos modelos MOM: todos eles consistem em um ou mais estados de destino e quaisquer estados de prefixo devem preceder os destinos, os estados de sufixo devem seguir os de destino, e outros estados devem ficar em segundo plano. Essa estrutura facilita a aprendizagem de MOMs a partir de exemplos. Com uma estrutura parcialmente especificada, o algoritmo para a frente e para trás pode ser utilizado para aprender tanto as probabilidades de transição $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1})$ entre estados como o modelo de observação, $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$, que informa qual a probabilidade de cada palavra estar em cada estado. Por exemplo, a palavra "sexta-feira" teria alta probabilidade de estar em um ou mais dos estados de destino do MOM e menor probabilidade de estar em outros lugares.

Com dados de treinamento suficientes, o MOM aprende automaticamente uma estrutura de datas que achamos intuitiva: a data do MOM pode ter um estado de destino em que a probabilidade alta de palavras são "segunda-feira", "terça-feira" etc., e que tem alta probabilidade de transição para um estado de destino com as palavras "Jan", "Janeiro", "Fev" etc. A Figura 22.2 mostra o MOM para o locutor de um comunicado, como aprendido dos dados. O prefixo abrange expressões como "locutor" e "seminário por", e o destino tem um estado que abrange títulos e primeiros nomes e outro estado que abrange iniciais e sobrenomes.

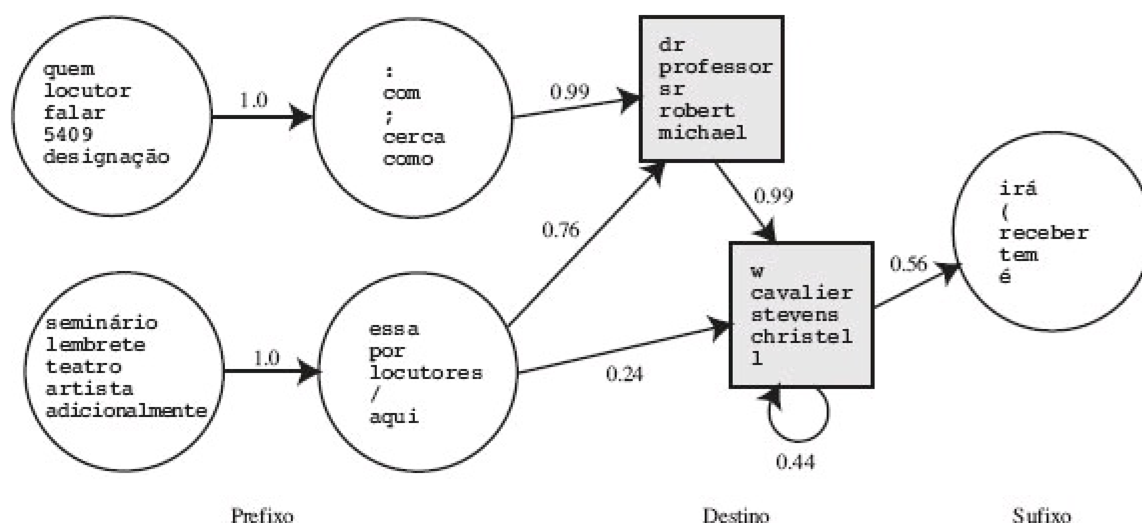


Figura 22.2 Modelo oculto de Markov para o **locutor** de um comunicado. Os dois estados nos quadrados são o destino (observe que o segundo estado de destino tem um autolaço e, assim, o destino pode corresponder a uma sequência de qualquer tamanho), os quatro círculos para a esquerda são os prefixos, e aquele da direita é o sufixo. Para cada estado são mostradas apenas algumas das palavras de alta probabilidade. De Freitag e McCallum (2000).

Uma vez que os MOMs foram informados, podemos aplicá-los a um texto usando o algoritmo Viterbi para encontrar o caminho mais provável pelos estados do MOM. Uma abordagem é a aplicação de cada atributo MOM separadamente; nesse caso seria de esperar que os MOMs gastassem a maior parte do tempo nos estados em segundo plano. Isso é apropriado quando a extração é esparsa — quando o número de palavras extraído é pequeno em comparação com o comprimento do texto.

Outra abordagem é combinar todos os atributos individuais em um MOM grande que, então,

encontraria um caminho que vagueia através de atributos de destino diferentes, primeiro encontrando um locutor destino, depois uma data destino etc. MOMs separados são melhores quando esperamos apenas um de cada atributo em um texto, e um MOM grande é melhor quando os textos têm o formato mais livre e denso com atributos. Com qualquer uma das abordagens, ao final temos uma coleção de observações de atributos de destino e precisamos decidir o que fazer com eles. Se todos os atributos esperados têm um preenchedor de destino, a decisão é fácil: temos um exemplo da relação desejada. Se houver múltiplos preenchedores, precisamos decidir qual escolher, como discutimos com o modelo baseado em sistemas. Os MOMs têm a vantagem de fornecer números de probabilidade que podem ajudar a fazer a escolha. Se alguns destinos estiverem faltando, precisamos decidir se essa é uma instância da relação desejada a todos ou se os destinos encontrados são falsos positivos. Um algoritmo de máquina de aprendizagem pode ser treinado para fazer essa escolha.

22.4.3 Campos aleatórios condicionais para a extração de informações

Um problema com os MOMs para a tarefa de extração de informações é que ele modela uma porção de probabilidades que nós realmente não precisamos. Um MOM é um modelo gerativo, que modela o conjunto completo de probabilidade de observações e estados ocultos, e, portanto, pode ser usado para gerar amostras. Isto é, podemos usar o modelo MOM não só para analisar um texto e recuperar o locutor e a data, mas também para gerar uma instância aleatória de um texto contendo um locutor e uma data. Uma vez que não estamos interessados nessa tarefa, é natural perguntar se poderíamos estar melhor com um modelo que não se incomodasse em modelar essa possibilidade. Tudo o que precisamos para compreender um texto é um **modelo discriminativo**, que modela a probabilidade condicional dos atributos ocultos dadas as observações (o texto). Dado um texto $\mathbf{e}_{1:n}$, o modelo condicional encontra a sequência de estado oculto $\mathbf{X}_{1:n}$ que maximiza $P(\mathbf{X}_{1:n} \mid \mathbf{e}_{1:n})$.

Modelar isso diretamente nos dá alguma liberdade. Não precisamos dos pressupostos da independência do modelo de Markov — podemos ter um $\mathbf{x}_t$ que seja dependente de $\mathbf{x}_1$. Uma estrutura conceitual para esse tipo de modelo é o **campo aleatório condicional**, ou CAC, que modela uma distribuição de probabilidade condicional de um conjunto de variáveis destino dado um conjunto de variáveis observadas. Como as redes bayesianas, os CACs podem representar muitas estruturas diferentes de dependências entre as variáveis. Uma estrutura comum é o **campo aleatório condicional de cadeia linear** representando dependências de Markov entre as variáveis em uma sequência temporal. Assim, os MOMs são a versão temporal dos modelos de Bayes ingênuos, e as CACs de cadeia linear são a versão temporal da regressão logística, onde o destino previsto é uma sequência de estado inteira, em vez de uma única variável binária.

Seja $\mathbf{e}_{1:n}$ as observações (por exemplo, palavras em um documento), e $\mathbf{x}_{1:n}$ a sequência de estados ocultos (por exemplo, os estados prefixo, destino e sufixo). O campo aleatório condicional de cadeia linear define uma distribuição de probabilidade condicional:

$$\mathbf{P}(\mathbf{x}_{1:N}|\mathbf{e}_{1:N}) = \alpha\, e^{\left[\sum_{i=1}^{N} F(\mathbf{x}_{i-1},\mathbf{x}_i,\mathbf{e},i)\right]},$$

onde α é um fator de normalização (para certificar-se de que a soma das probabilidades é 1) e F é

uma característica definida como a soma ponderada de um conjunto de k funções características de componentes:

$$F(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{e}, i) = \sum_k \lambda_k f_k(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{e}, i) .$$

Os valores do parâmetro λ_k são aprendidos com um procedimento de estimativa MAP (máximo *a posteriori*) que maximiza a probabilidade condicional de dados de treinamento. As funções características são os principais componentes de um CAC. A função f_k tem acesso a um par de estados adjacentes, x_{i-1} e x_i, mas também à sequência de observação inteira (palavra) **e** e à posição atual na sequência temporal, i. Isso nos dá grande flexibilidade na definição das características. Podemos definir uma simples função característica, por exemplo, a que que produz um valor de 1 se a palavra atual for ANDREW e o estado atual for LOCUTOR:

$$f_1(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{e}, i) = \begin{cases} 1 \text{ se } \mathbf{x}_i = \text{LOCUTOR e } \mathbf{e}_i = \text{ANDREW} \\ 0 \text{ caso contrário} \end{cases}$$

Como características como essas são utilizadas? Depende de seus pesos correspondentes. Se $\lambda_1 > 0$, então sempre que f_1 for verdadeiro, aumenta a probabilidade da sequência do estado oculto $\mathbf{x}_{1:n}$. Isso é outra maneira de dizer "o modelo CAC deveria dar preferência as estado alvo LOCUTOR para o termo ANDREW". Se, por outro lado $\lambda < 0$, o modelo CAC tentará evitar essa associação e, se $\lambda_1 = 0$, essa característica será ignorada. Os valores dos parâmetros podem ser inicializados manualmente ou ser pegos dos dados. Agora, considere uma segunda função característica:

$$f_2(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{e}, i) = \begin{cases} 1 \text{ se } \mathbf{x}_i = \text{LOCUTOR e } \mathbf{e}_{i+1} = \text{DISSE} \\ 0 \text{ caso contrário.} \end{cases}$$

Esse recurso é verdadeiro se o estado atual for LOCUTOR e a palavra seguinte for "disse". Alguém poderia, portanto, esperar um valor de λ_2 positivo com a característica. Observe que o mais interessante é que tanto f_1 como f_2 são válidos ao mesmo tempo para uma sentença como "Andrew disse…". Nesse caso, as duas características se sobrepõem e ambas impulsionam a crença de que $\mathbf{x}_1$ = LOCUTOR. Por causa da suposição de independência, os MOMs não podem usar recursos sobrepostos; os CACs podem. Além disso, uma característica em CAC é que ela pode usar qualquer parte da sequência $\mathbf{e}_{1:n}$. As características também podem ser definidas ao longo das transições entre os estados. As características que foram definidas aqui eram binárias, mas em geral uma função característica pode ser qualquer função de valor real. Para domínios em que temos algum conhecimento sobre os tipos de características que gostaríamos de incluir, o formalismo CAC nos oferece grande flexibilidade na definição deles. Essa flexibilidade pode levar a precisões que são maiores com modelos mais flexíveis, tais como MOMs.

22.4.4 Extração de ontologias a partir de *corpora* grandes

Até agora pensamos em extração de informações, como encontrar um conjunto de relações específicas (por exemplo, locutor, hora, local) em um texto específico (por exemplo, um comunicado). Uma aplicação diferente da tecnologia de extração é a construção de uma base ampla de conhecimento ou ontologia de fatos de um *corpus*. Isso é diferente por três razões: primeiro, é em aberto — queremos adquirir fatos sobre todos os tipos de domínios, não apenas de um domínio específico. Em segundo lugar, com um *corpus* grande, essa tarefa é dominada pela precisão, não revocação, exatamente como perguntas e respostas na Web (Seção 22.3.6). Terceiro, os resultados podem ser agregados estatísticos recolhidos de várias fontes, em vez de ser extraídos de um texto específico.

Por exemplo, Hearst (1992) examinou o problema de aprender uma ontologia de categorias e subcategorias de conceito de um *corpus* grande. (Em 1992, um *corpus* grande era uma enciclopédia com mil páginas; hoje seria um *corpus* de 100 milhões de páginas da Web.) O trabalho concentrou-se em modelos que são muito gerais (não vinculados a um domínio específico) e tinham alta precisão (eram quase sempre corretos quando correspondiam), mas com baixa cobertura (nem sempre coincidiam). Aqui está um dos modelos mais produtivos:

$$NP\ \textbf{tal que}\ NP(,\ NP)^*\ (,)?\ ((\textbf{e}\mid\textbf{ou})\ NP)?\ .$$

Aqui as palavras em negrito e vírgulas devem aparecer literalmente no texto, mas os parênteses são do agrupamento, o asterisco significa *repetição de zero ou mais* e o ponto de interrogação significa *opcional*. *NP* é uma variável que suporta um sintagma nominal; o Capítulo 23 descreve como identificar sintagmas nominais, por ora apenas assuma que sabemos que algumas palavras são substantivos e outras palavras (tais como verbos) podemos assumir confiantemente que não são parte de um sintagma nominal simples. Esse modelo corresponde ao texto "doenças como a raiva afetam o seu cão" e "suporta protocolos de rede tais como DNS", concluindo que a raiva é uma doença e DNS é um protocolo de rede. Pode-se construir modelos semelhantes com as palavras-chave "inclusive", "especialmente" e "ou outros". Certamente esses modelos vão deixar de coincidir com muitas passagens relevantes, como "A raiva é uma doença". Isso é intencional. O modelo "*NP* é um *NP*" de fato, por vezes, denota uma relação de subcategoria, mas muitas vezes significa outra coisa, como em "Há um Deus" ou "Ela está um pouco cansada". Com um *corpus* grande podemos nos dar ao luxo de ser exigentes; usar apenas os modelos de alta precisão. Vamos perder muitas declarações de um relacionamento de subcategoria, mas muito provavelmente encontraremos uma paráfrase da declaração em outro lugar no *corpus* de uma forma que possamos usar.

22.4.5 Construção automatizada do padrão

A relação de *subcategoria* é tão fundamental que vale a pena fazer manualmente alguns padrões para ajudar a identificar os casos de ocorrências no texto em linguagem natural. Mas o que dizer dos milhares de outras relações no mundo? Criar e depurar padrões para todos eles, tomando como base alunos de pós-graduação de IA pelo mundo, não é suficiente. Felizmente, é possível *aprender* padrões a partir de alguns exemplos e então usá-los para aprender mais exemplos, de onde se pode aprender mais padrões, e assim por diante. Em uma das primeiras experiências desse tipo, Brin

(1999) começou com um conjunto de dados de apenas cinco exemplos:

(“Isaac Asimov”, “The Robots of Dawn”)
(“David Brin”, “Startide Rising”)
(“James Gleick”, “Chaos-Making a New Science”)
(“Charles Dickens”, “Great Expectations”)
(“William Shakespeare”, “The Comedy of Errors”)

Claramente, esses são exemplos da relação autor-título, mas o sistema de aprendizagem não tem conhecimento de autores ou títulos. As palavras nesses exemplos foram utilizadas em uma pesquisa em um *corpus* na Web, resultando em 199 combinações. Cada combinação é definida como tuplas de sete sequências,

(Autor, Título, Ordem, Prefixo, Meio, Sufixo, URL),

onde *Ordem* será verdadeiro se o autor vier primeiro e falso se o título vier primeiro, *Meio* são os caracteres entre o autor e título, *Prefixo* são os 10 caracteres antes da correspondência, *Sufixo* são os 10 caracteres após a correspondência e *URL* é o endereço da Web onde a correspondência foi feita.

Dado um conjunto de correspondências, um esquema simples de geração de padrões pode encontrar padrões para explicar as correspondências. A linguagem dos padrões foi projetada para ter um mapeamento junto às correspondências, para ser passível de aprendizagem automatizada e para enfatizar a alta precisão (possivelmente com o risco de menor cobertura). Cada padrão tem os mesmos sete componentes como correspondência. O *Autor* e o *Título* são expressões regulares consistindo em quaisquer caracteres (mas começando e terminando em letras) e restritos a ter um comprimento de metade do comprimento mínimo dos exemplos até duas vezes o comprimento máximo. Prefixo, meio e sufixo são restritos a sequências literais, que não sejam expressões regulares. O meio é o mais fácil de aprender: cada sequência de meio distinta em um conjunto de correspondências é um modelo de candidato distinto. Para tal candidato, o padrão de *prefixo* é então definido como o sufixo comum mais longo de todos os prefixos nas correspondências, e o *sufixo* é definido como o prefixo comum mais longo de todos os sufixos nas correspondências. Se qualquer um deles for de comprimento zero, o padrão será rejeitado. A *URL* do padrão é definida como o prefixo mais longo das URLs nas correspondências.

No experimento executado por Brin, as primeiros 199 combinações geraram três padrões. O padrão mais produtivo foi

<LI> <B> *Título* </ B> por *Autor* (
URL: www.sff.net/locus/c

Os três padrões foram então usados para recuperar mais 4.047 exemplos (autor, título). Os exemplos foram usados para gerar mais padrões, e assim por diante, acabando por produzir mais de 15.000 títulos. Dado um bom conjunto de padrões, o sistema pode coletar um bom conjunto de exemplos. Dado um bom conjunto de exemplos, o sistema pode construir um bom conjunto de padrões.

A maior fraqueza dessa abordagem é a sensibilidade ao ruído. Se um dos primeiros poucos

padrões estiver incorreto, os erros podem se propagar rapidamente. Uma forma de limitar esse problema é não aceitar um novo exemplo a menos que seja verificado por vários padrões, e não aceitar um novo padrão, a menos que descubra vários exemplos que são também encontrados por outros padrões.

22.4.6 Leitura de máquina

A construção de padrões automatizados é um grande passo para a construção do padrão artesanal, mas ainda requer um punhado de exemplos rotulados de cada relação para começar. Para construir uma grande ontologia com muitos milhares de relações, mesmo essa quantidade de trabalho seria onerosa; gostaríamos de ter um sistema de extração *sem* entrada humana de espécie nenhuma — um sistema que pudesse ler por conta própria e construísse sua própria base de dados. Tal sistema seria independente da relação; trabalharia para qualquer relação. Na prática, esses sistemas trabalham com *todas* as relações em paralelo, devido às exigências de E/S de *corpora* de grande dimensão. Eles se comportam menos como um sistema de extração de informações tradicional que se destina a poucas relações e mais como um leitor humano que aprende do próprio texto; por esse motivo, o campo tem sido chamado de **leitura de máquina**.

Um sistema de leitura de máquina representativo é o TEXTRUNNER (Banko e Etzioni, 2008). O TEXTRUNNER utiliza cotreinamento para aumentar seu desempenho, mas precisa de algo para conseguir desenvolver-se por si mesmo. No caso de Hearst (1992), padrões específicos (por exemplo, *tais como*) forneceram o reinicialização e, para Brin (1998), foi um conjunto de cinco pares de autor-título. Para o TEXTRUNNER, a inspiração original foi uma taxonomia de oito modelos sintáticos muito gerais, como mostrado na Figura 22.3. Sentiu-se que um pequeno número de padrões como esse poderia abranger a maior parte das formas como as relações são expressas em inglês. O processo é acelerado a partir de um conjunto de exemplos rotulados e extraídos do Penn Treebank, um *corpus* de sentenças analisadas. Por exemplo, da análise da sentença “Einstein recebeu o Prêmio Nobel em 1921”, o TEXTRUNNER é capaz de extrair a relação (“Einstein”, “recebeu”, “Prêmio Nobel”).

Dado um conjunto de exemplos rotulados desse tipo, o TEXTRUNNER treina uma cadeia linear CAC para extrair mais exemplos de textos sem rótulo. As características no CAC incluem palavras funcionais como “para” e “de” e “o”, mas não substantivos e verbos (e não frases nominais ou verbais). Como o TEXTRUNNER é independente de domínio, ele não pode confiar em listas predefinidas de substantivos e verbos.

Tipo	Modelo	Exemplo	Frequência
Verbo	NP_1 *Verbo* NP_2	X estabeleceu Y	38%
Substantivo-Prep	NP_1 *NP Prep* NP_2	X acordou com Y	23%
Verbo-Prep	NP_1 *Verbo Prep* NP_2	X se moveu para Y	16%
Infinitivo	NP_1 **para** *Verbo* NP_2	X planeja adquirir Y	9%
Modificador	NP_1 *Verbo* NP_2 *Subst*	X é vencedor de Y	5%
Subst-composto		X-Y tratar	2%

Verbo-composto Aposto	NP_1 (, | **e** | **-** |:) NP_2 NP NP_1 (, | **e**) NP_2 *verbo* NP_1 NP (: |,)? NP_2	X, Y fundir X cidade natal : Y	1% 1%

Figura 22.3 Oito modelos gerais que cobrem cerca de 95% das formas como as relações são expressas em inglês.

O TEXTRUNNER atinge precisão de 88% e cobertura de 45% (F_1 de 60%) em um grande *corpus* da Web. O TEXTRUNNER extraiu centenas de milhões de fatos a partir de um *corpus* de meio bilhão de páginas da Web. Por exemplo, mesmo não tendo conhecimentos médicos predefinidos, extraiu cerca de 2.000 respostas para a consulta [o que mata as bactérias]; respostas corretas incluem antibióticos, ozônio, cloro, Cipro e brotos de brócolis. As respostas duvidosas incluem "água", que veio da sentença "água fervendo por pelo menos 10 minutos vai matar a bactéria". Seria melhor atribuir isso a "água fervendo" em vez de apenas a "água".

Com as técnicas delineadas neste capítulo e novas invenções contínuas, estamos começando a chegar mais perto do objetivo da leitura automática de máquina.

22.5 RESUMO

Os principais pontos deste capítulo são os seguintes:

- Modelos de linguagem probabilística com base em *n*-gramas recuperam uma quantidade surpreendente de informação sobre um idioma. Eles podem ter bom desempenho em tarefas tão diversas como identificação da linguagem, correção ortográfica, classificação do gênero e reconhecimento do nome da entidade.
- Esses modelos de linguagem podem ter milhões de características, então é importante a seleção das características e o pré-processamento dos dados para reduzir o ruído.
- A **classificação de texto** pode ser feita com modelos de *n*-gramas de Bayes ingênuo ou com qualquer um dos algoritmos de classificação que discutimos anteriormente. A classificação também pode ser vista como um problema na compressão de dados.
- Sistemas de **recuperação de informação** utilizam um modelo de linguagem muito simples, baseado em sacos de palavras, mas ainda conseguem bom desempenho em termos de **cobertura** e **precisão** em *corpora* muito grandes de texto. Em *corpora* da Web, algoritmos de análise de link melhoram o desempenho.
- **Perguntas e respostas** podem ser tratadas por uma abordagem baseada na recuperação de informação, para questões que têm múltiplas respostas no *corpus*. Quando estiver disponível no *corpus* mais respostas, podemos utilizar técnicas que enfatizam a precisão, em vez da cobertura.
- Sistemas de **extração de informação** utilizam um modelo mais complexo que inclui noções limitadas de sintaxe e semântica na forma de modelos. Podem ser construídos a partir de autômatos de estado finito, MOMs ou campos aleatórios condicionais e ser aprendidos de exemplos.

- Na construção de um sistema de linguagem estatística, o melhor é criar um modelo que possa fazer bom uso dos **dados** disponíveis, mesmo que o modelo pareça demasiado simplista.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Markov (1913) propôs modelos de letra *n*-grama para a modelagem de linguagem. Claude Shannon (Shannon e Weaver, 1949) foram os primeiros a gerar modelos de *n*-gramas de palavras em inglês. Chomsky (1956, 1957) apontou as limitações dos modelos de estado finito comparados com modelos de contexto livre, concluindo que "modelos probabilísticos não dão nenhuma visão particular em alguns dos problemas básicos da estrutura sintática". Isso é verdade, mas modelos probabilísticos fornecem discernimento em alguns outros problemas básicos — problemas que os modelos de contexto livre ignoram. As observações de Chomsky tiveram o efeito infeliz de assustar muitas pessoas acerca dos modelos estatísticos por duas décadas, até que esses modelos ressurgiram para uso em reconhecimento de fala (Jelinek, 1976).

Kessler *et al*. (1997) mostraram como aplicar modelos de *n*-grama de caractere para classificação de gênero, e Klein *et al*. (2003) descreveram o reconhecimento de nome da entidade com modelos de caracteres. Franz e Brants (2006) descreveram o *corpus* *n*-grama do Google de 13 milhões de palavras únicas de um trilhão de palavras de texto da Web; está agora publicamente disponível. O modelo de **saco de palavras** recebeu seu nome de uma passagem do linguista Zellig Harris (1954), "a linguagem não é apenas um saco de palavras, mas uma ferramenta com propriedades particulares". Norvig (2009) fornece alguns exemplos de tarefas que podem ser realizadas com modelos de *n*-grama.

Alisamento com adição de um, primeiro sugerido por Pierre-Simon Laplace (1816), foi formalizado por Jeffreys (1948), e alisamento de interpolação foi devido a Jelinek e Mercer (1980), que o utilizaram para reconhecimento de voz. Outras técnicas incluem alisamento de Witten-Bell (1991), alisamento de Good-Turing (Church e Gale, 1991) e alisamento de Kneser-Ney (1995), técnicas de alisamento de pesquisa de Chen e Goodman (1996) e Goodman (2001).

Modelos de *n*-grama simples de letra e palavra não são os únicos modelos probabilísticos possíveis. Blei *et al.* (2001) descreveram um modelo de texto probabilístico chamado **alocação latente de Dirichlet** que visualiza um documento como uma mistura de temas, cada um com sua própria distribuição de palavras. Esse modelo pode ser visto como extensão e racionalização do modelo de **indexação semântica latente** de (Deerwester *et al*., 1990) (ver também Papadimitriou *et al*. (1998)) e também está relacionado a um modelo de mistura de múltiplas causas (Sahami *et al*., 1996).

Manning e Schütze (1999) e Sebastiani (2002) pesquisaram técnicas de classificação de texto. Joachims (2001) usaram a teoria de aprendizagem estatística e máquinas de vetor de suporte para fornecer uma análise teórica de quando a classificação será bem-sucedida. Apté *et al.* (1994) relataram precisão de 96% na classificação de artigos de notícias de Reuters na categoria "Lucro". Koller e Sahami (1997) relataram precisão no relatório de até 95% com um classificador de Bayes ingênuo e até 98,6% com um classificador de Bayes que é a razão das dependências entre as características. Lewis (1998) pesquisou por quarenta anos a aplicação de técnicas de Bayes ingênuo

para classificação e recuperação de texto. Schapire e Singer (2000) mostraram que classificadores lineares simples podem muitas vezes alcançar precisão tão boa quanto os modelos mais complexos e são mais eficientes para avaliar. Nigam *et al*. (2000) mostraram como usar o algoritmo EM para rotular documentos sem rótulo, aprendendo assim um modelo melhor de classificação. Witten *et al*. (1999) descreveram algoritmos de compressão para a classificação e mostraram a ligação profunda entre o algoritmo de compressão LZW e os modelos de linguagem de máxima entropia.

Muitas das técnicas do modelo de *n*-gramas também são usadas em problemas de bioinformática. Bioestatística e PNL probabilística estão chegando mais próximo por lidar com sequências longas, estruturadas, escolhidas de um alfabeto de constituintes.

O campo de **recuperação de informação** está experimentando um novo crescimento de interesse, provocado pela ampla utilização de busca na Internet. Robertson (1977) deu uma visão geral precoce e introduziu o princípio da equiparação da probabilidade. Croft *et al*. (2009) e Manning *et al.* (2008) foram os primeiros livros didáticos a abranger a busca baseada na Web, bem como RI tradicional. Hearst (2009) abrangeu interfaces de usuário para buscas na Web. A conferência TREC, organizada pelo National Institute of Standards and Technology (NIST), organizou uma competição anual de sistemas de IR e publicou procedimentos com resultados. Nos primeiros sete anos de competição, o desempenho praticamente dobrou.

O modelo mais popular para RI é o **modelo de espaço vetorial** (Salton *et al*., 1975). O trabalho de Salton dominou os primeiros anos do campo. Existem duas alternativas de modelos probabilísticos, um por Ponte e Croft (1998) e um por Maron e Kuhns (1960) e Robertson e Sparck Jones (1976). Lafferty e Zhai (2001) mostram que os modelos são baseados na mesma distribuição de probabilidade conjunta, mas que a escolha do modelo tem implicações para o treinamento dos parâmetros. Craswell *et al.* (2005) descreveram a função de pontuação BM25 e Svore e Burges (2009) descreveram como o BM25 pode ser melhorado com uma abordagem de aprendizagem de máquina que incorpora clique em dados — exemplos de busca de consultas passadas e os resultados que foram clicados.

Brin e Page (1998) descreveram o algoritmo PageRank e a implementação de um mecanismo de busca na Web. Kleinberg (1999) descreveu o algoritmo HITS. Silverstein *et al*. (1998) investigaram um log de um bilhão de buscas na Web. O periódico *Information Retrieval* e os procedimentos da conferência anual *SIGIR* abrangem desenvolvimentos recentes no campo.

Os programas de extração de informações iniciais incluem GUS (Bobrow *et al*., 1977) e Frump (DeJong, 1982). A Message Understand Conferences (MUC), patrocinada pelo governo dos Estados Unidos, tem incentivado a extração de informações recentes. O sistema de estado finito, FASTUS, foi desenvolvido por Hobbs *et al*. (1997). Foi baseado em parte na ideia de Pereira e Wright (1991) de usar FSAs como aproximações a gramáticas de estrutura de frase. Roche e Schabes (1997), Appelt (1999) e Muslea (1999) ofereceram pesquisas de sistemas baseados em modelos. Craven *et al*. (2000), Pasca *et al*. (2006), Mitchell (2007) e Durme e Pasca (2008) extraíram grandes bancos de dados de fatos.

Freitag e McCallum (2000) discutiram MOMs para extração de informação. As CAcs foram introduzidas por Lafferty *et al.* (2001); um exemplo de sua utilização para extração de informações é descrito em McCallum (2003) e um tutorial com orientação prática foi dado por Sutton e McCallum (2007). Sarawagi (2007) forneceu um panorama abrangente.

Banko *et al*. (2002) apresentaram o sistema ASKMSR de perguntas e respostas; um sistema semelhante é devido a Kwok *et al*. (2001). Pasca e Harabagiu (2001) discutiram um sistema de pergunta e resposta de dispusta. Riloff (1993) apresentou duas antigas abordagens influentes para a engenharia do conhecimento automatizado que mostrou que um dicionário construído automaticamente desempenha quase tão bem como um dicionário de domínio específico feito manualmente com todo o cuidado. Yarowsky (1995) mostrou que a tarefa de classificação do sentido da palavra poderia ser realizado por meio de treinamento não supervisionado em um *corpus* de texto sem rótulo com precisão tão boa quanto com métodos supervisionados.

A ideia de extrair modelos e exemplos simultaneamente de um punhado de exemplos rotulados foi desenvolvida de forma independente e simultaneamente por Blum e Mitchell (1998), que o chamou de **cotreinamento**, e Brin (1998), que o chamou de DIPRE (Dual Iterative Pattern Relational Extraction, ou seja, extração de relação do padrão iterativo duplo). Você pode ver por que aderiram ao termo *cotreinamento*. Um trabalho anterior similar, sob o nome de bootstrapping, ou seja, algo que se desenvolve por si só, foi feito por Jones *et al*. (1999). O método avançou pelos sistemas QXTRACT (Agichtein e Gravano, 2003) e KnowItAll (Etzioni *et al*., 2005). A leitura de máquina foi introduzida por Mitchell (2005) e Etzioni *et al*. (2006), e é o foco do projeto TEXTRUNNER (Banko *et al*., 2007; Banko e Etzioni, 2008).

Este capítulo foi centrado no texto de linguagem natural, mas também é possível fazer extração de informação com base na estrutura física ou no leiaute do texto e não na estrutura linguística. As listas HTML e tabelas em bancos de dados HTML e relacional são o lar para dados que podem ser extraídos e consolidados (Hurst, 2000; Pinto *et al*., 2003; Cafarella *et al*., 2008).

A Association for Computacional Linguistics (ACL) realiza conferências regulares e publica a revista *Computational Linguistics*. Há também uma Conferência Internacional em Linguística Computacional (COLING). O livro de Manning e Schütze (1999) cobre processamento da linguagem estatística, enquanto Jurafsky e Martin (2008) dão uma introdução abrangente ao discurso e processamento de linguagem natural.

EXERCÍCIOS

22.1 Esse exercício explora a qualidade do modelo de *n*-grama da linguagem. Encontre ou crie um corpus monolíngue de 100.000 palavras ou mais. Divida-o em palavras e calcule a frequência de cada palavra. Quantas palavras distintas existem? Conte também a frequência dos bigramas (duas palavras consecutivas) e trigramas (três palavras consecutivas). Agora use essas frequências para gerar a linguagem: a partir dos modelos unigrama, bigrama e trigrama, gere um texto de 100 palavras por vez, fazendo escolhas randômicas de acordo com a frequência de contagem. Compare os três textos gerados com a linguagem real. Finalmente, calcule a perplexidade de cada modelo.

22.2 Escreva um programa para fazer **segmentação** de palavras sem espaços. Dada uma sequência, tal como a URL "thelongestlistofthelongeststuffatthelongestdomainnameatlonglast.com", retorne uma lista de palavras componentes: ["the", "longest","list"…]. Essa tarefa é útil para a

análise de URLs, para a correção de ortografia quando as palavras se unem e para idiomas como o chinês, que não tem espaços entre as palavras. Pode ser resolvido com um modelo de palavra unigrama ou bigrama e um algoritmo de programação dinâmica semelhante ao algoritmo de Viterbi.

22.3 (Adaptado de Jurafsky e Martin, 2000.) Neste exercício, você vai desenvolver um classificador de autoria: dado um texto, o classificador prediz qual dos dois autores candidatos escreveu o texto. Obtenha amostras de texto de dois autores diferentes. Separe-os em conjuntos de treinamento e de teste. Agora treine um modelo de linguagem no conjunto de treinamento. Você pode escolher que recursos utilizar; *n*-gramas de palavras ou letras são os mais fáceis, mas você pode adicionar recursos que ache que pode ajudar. Em seguida, calcule a probabilidade do texto sob cada modelo de linguagem e escolha o modelo mais provável. Avalie a precisão da técnica. Como é que a precisão muda à medida que você altera o conjunto de características? Esse subcampo da linguística é chamado de **estilometria**; seu sucesso inclui a identificação do autor do disputado *Federalist Papers* (Mosteller e Wallace, 1964) e algumas obras de Shakespeare disputadas (Hope, 1994). Khmelev e Tweedie (2001) produziram bons resultados com um modelo bigrama simples de letras.

22.4 Este exercício trata da classificação de spam de e-mail. Crie um *corpus* de spam de e-mail e um de não spam de e-mail. Examine cada *corpus* e decida quais características parecem ser úteis para a classificação: palavras unigrama? bigramas? tamanho da mensagem, remetente, tempo de chegada? Depois treine um algoritmo de classificação (árvore de decisão, Bayes ingênuo, SVM, regressão logística ou algum outro algoritmo de sua escolha) em um conjunto de treinamento e relate a sua precisão em um conjunto de teste.

22.5 Crie um conjunto de teste de dez consultas e coloque-as nos três principais mecanismos de busca da Web. Avalie cada um para uma precisão de 1, 3 e 10 documentos. Você pode explicar as diferenças entre os mecanismos?

22.6 Tente determinar qual dos mecanismos de busca do exercício anterior estão usando sensibilidade à caixa alta, derivação, sinônimo*s,* correção ortográfica?

22.7 Escreva uma expressão regular ou um pequeno programa para extrair nomes de empresas. Teste-o em um *corpus* de artigos de notícias de negócios. Relate a revocação e a precisão.

22.8 Considere o problema de tentar avaliar a qualidade de um sistema de RI que retorna uma lista ordenada de respostas (como a maioria dos mecanismos de busca da Web). A medida de qualidade adequada depende do modelo presumido que a pesquisadora está tentando atingir e de qual estratégia emprega. Para cada um dos seguintes modelos, proponha uma medida correspondente numérica.

a. A pesquisadora vai examinar as primeiras vinte respostas retornadas, com o objetivo de obter o máximo de informações relevantes possível.

b. A pesquisadora precisa de apenas um documento relevante e examina a lista até encontrar o primeiro.

c. A pesquisadora tem uma consulta bastante pequena e é capaz de analisar todas as respostas obtidas. Ela quer ter certeza de que já viu tudo na coleção do documento que é relevante para a sua consulta. (Por exemplo, uma advogada quer ter certeza de que tenha encontrado *todos os*

precedentes relevantes e está disposta a gastar recursos consideráveis nisso.)

d. A pesquisadora precisa apenas de um documento relevante para a consulta e pode se dar ao luxo de pagar um assistente de pesquisa por uma hora de trabalho para examinar os resultados. O assistente pode examinar 100 documentos recuperados em uma hora. O assistente vai cobrar do pesquisador pela hora cheia, independentemente de encontrá-lo imediatamente ou ao final da hora.

e. A pesquisadora vai procurar por todas as respostas. Examinar um documento custa $\$A$; encontrar um documento relevante custa $\$B$; não conseguir encontrar um documento relevante custa $\$C$ para cada documento relevante não encontrado.

f. A pesquisadora quer reunir tanto documentos relevantes quanto possível, mas precisa de constante encorajamento. Ela examina os documentos em ordem. Se os documentos que ela examinou até agora são bons, vai continuar; caso contrário, vai desistir.

---

[1] Possivelmente com a exceção do trabalho pioneiro de T. Geisel (1955).

[2] Denotamos uma consulta de pesquisa como [*consulta*]. Colchetes são usados em vez de aspas para que possamos distinguir a consulta [“duas palavras”] de [duas palavras].

[3] O nome se refere tanto a páginas da Web como ao seu co-autor Larry Page (Brin e Page, 1998).

CAPÍTULO 23

Linguagem natural para comunicação

Em que vemos como os seres humanos se comunicam uns com os outros em linguagem natural e como os agentes de computador poderiam se juntar à conversa.

A **comunicação** é a troca intencional de informações provocada pela produção e percepção de **sinais** extraídos de um sistema compartilhado de sinais convencionais. A maioria dos animais utiliza sinais para representar mensagens importantes: comida aqui, predador por perto, abordagem, retirada, vamos acasalar. Em um mundo parcialmente observável, a comunicação pode ajudar os agentes a terem sucesso porque eles podem aprender informações que serão observadas ou deduzidas por outros.

Os seres humanos são os mais conversadores de todas as espécies e, para os agentes de computador tornarem-se úteis, eles precisam aprender a falar a linguagem. Neste capítulo, examinaremos os modelos de linguagem para comunicação. Os modelos que visam à compreensão profunda de uma conversa precisam ser necessariamente mais complexos que os modelos simples que visam, por exemplo, à classificação de spam. Começaremos com modelos gramaticais da estrutura frasal das sentenças, adicionaremos semântica ao modelo e depois o aplicaremos para a tradução automática e para o reconhecimento de fala.

23.1 GRAMÁTICAS DE ESTRUTURA FRASAL

Os modelos de linguagem de *n*-grama do Capítulo 22 foram baseados em sequências de palavras. O grande problema desses modelos é a **dispersão de dados** — com um vocabulário de, digamos, 10^5 palavras, existem 10^{15} probabilidades de trigramas a estimar e, assim, um *corpus* de até um trilhão de palavras não será capaz de fornecer estimativas confiáveis para todos eles. Podemos resolver o problema da dispersão através da generalização. Do fato de que, em inglês, "*black dog*" é mais frequente do que "*dog black*", e de observações similares, podemos formar a generalização de que os adjetivos tendem a vir antes dos substantivos em inglês (enquanto tendem a seguir os substantivos em francês: "*chien noir*" é mais frequente).

Claro que há sempre exceções: "*galore*" é um adjetivo que segue o substantivo que ele modifica. Apesar das exceções, a noção de **categoria lexical** (também conhecida como **parte do discurso**),

como substantivo ou adjetivo, é uma generalização útil — útil por si mesma, mas mais ainda quando juntamos categorias lexicais para formar **categorias sintáticas**, como o *sintagma nominal* ou *verbal,* e combinamos essas categorias sintáticas em árvores que representam a **estrutura frasal** das sentenças: sintagmas aninhados, cada um marcado com uma categoria.

CAPACIDADE GERATIVA

Os formalismos gramaticais podem ser classificados por sua **capacidade gerativa**: o conjunto de linguagens que eles podem representar. Chomsky (1957) descreve quatro classes de formalismos gramaticais que diferem apenas na forma das regras de reescrita. As classes podem ser organizadas em uma hierarquia, na qual cada classe pode ser utilizada para descrever todas as linguagens que também podem ser descritas por uma classe menos poderosa, bem como algumas linguagens adicionais. Aqui está a hierarquia, com a classe mais poderosa listada em primeiro lugar:

As gramáticas **recursivamente enumeráveis** utilizam regras irrestritas: ambos os lados das regras de reescrita podem ser qualquer número de símbolos de terminais e não terminais, como na regra $A\ B\ C \rightarrow D\ E$. Essas gramáticas são equivalentes a máquinas de Turing em seu poder de expressão.

As **gramáticas sensíveis ao contexto** são restritas apenas no fato de que o lado direito deve conter pelo menos tantos símbolos quanto o lado esquerdo. A designação “sensível ao contexto” resulta do fato de que uma regra como $A\ X\ B \rightarrow A\ Y\ B$ nos diz que um X pode ser reescrito como um Y no contexto de um A precedente e um B seguinte. As gramáticas sensíveis ao contexto podem representar linguagens como $a^n b^n c^n$ (uma sequência de n cópias de a, seguida pelo mesmo número de cópias de b e, depois, por igual número de cópias de c).

Em **gramáticas livres de contexto** (ou **GLC**s), o lado esquerdo consiste em um único símbolo de não terminal. Desse modo, cada regra define a reescrita do não terminal como o lado direito em *qualquer* contexto. GLCs são populares para gramáticas de linguagens naturais e linguagens de programação, embora hoje seja amplamente aceito que pelo menos algumas linguagens naturais têm construções que não são livres de contexto (Pullum, 1991). As gramáticas livres de contexto podem representar $a^n b^n$, mas não $a^n b^n c^n$.

As gramáticas **regulares** formam a classe mais restrita. Toda regra tem um único símbolo de não terminal no lado esquerdo e um símbolo de terminal opcionalmente seguido por um símbolo de não terminal no lado direito. As gramáticas regulares têm poder equivalente ao das máquinas de estados finitos. Elas são mal adaptadas às linguagens de programação porque não podem representar construções como abertura e fechamento de parênteses balanceados (uma variação da linguagem $a^n b^n$). O mais perto que elas podem chegar dessas linguagens é a representação de a^*b^*, uma sequência de qualquer número de símbolos a seguida por qualquer número de símbolos b.

As gramáticas posicionadas em ordem mais alta na hierarquia têm maior poder de expressão, mas os algoritmos para lidar com elas são menos eficientes. Até meados da década de 1980, os linguistas se concentravam em linguagens livres de contexto e sensíveis ao contexto. Desde então,

houve ênfase crescente em gramáticas regulares, provocada pela necessidade de processar e aprender de gigabytes e terabytes de texto on-line com muita rapidez, mesmo ao custo de uma análise menos completa. Como afirmou Fernando Pereira, "quanto mais velho fico, mais fundo desço na hierarquia de Chomsky". (Para entender o que ele quis dizer, compare Pereira e Warren (1980) com Mohri, Pereira e Riley (2002)) (e observe que os três autores trabalham agora em *corpora* de textos grandes no Google.)

Há muitos modelos de linguagem competindo baseados na ideia da estrutura de frase; descreveremos um modelo popular chamado de **gramática livre de contexto probabilística**, ou GLCP.[1] Uma **gramática** é um conjunto de regras que define uma **linguagem** como um conjunto de cadeias de palavras permitido. "Livre de contexto" é descrito nesta página e "probabilística" significa que a gramática atribui uma probabilidade a cada cadeia. Aqui está uma regra GLCP:

$$\begin{aligned} SV &\rightarrow \text{Verbo } [0{,}70] \\ &\mid \; SV\ SN\ [0{,}30]. \end{aligned}$$

Aqui SV (*sintagma verbal*) e SN (*sintagma nominal*) são **símbolos não terminais**. A gramática também se refere a palavras reais, que são chamadas de **símbolos terminais**. Essa regra está informando que, com probabilidade de 0,70 um sintagma verbal consiste unicamente em um verbo, e com probabilidade de 0,30 ele é um SV seguido por um SN. O Apêndice B descreve gramáticas livres de contexto não probabilísticas.

Vamos definir agora uma gramática para um pequeno fragmento de inglês adequado à comunicação entre os agentes que estão explorando o mundo de wumpus. Chamamos essa linguagem de ε_0. As seções posteriores melhorarão o ε_0 para torná-lo um pouco mais próximo do inglês real. É improvável que seja delineada uma gramática completa de inglês, mesmo porque não há duas pessoas que concordem inteiramente sobre o que constitui um inglês válido.

23.1.1 O léxico de ε_0

Primeiro, definimos o **léxico**, ou a lista de palavras permitidas. As palavras são agrupadas em categorias lexicais familiares aos usuários de dicionários: substantivos, pronomes e nomes próprios para denotar coisas, verbos para denotar eventos, adjetivos para modificar substantivos e advérbios para modificar verbos; e palavras funcionais: artigos (como *the*), preposições (*in*) e conjunções (*and*). A Figura 23.1 mostra um pequeno léxico para a linguagem ε_0.

Substantivo → **fedor** [0,05] | **brisa** [0,10] | **wumpus** [0,15] | **poços** [0,05] |. . .
Verbo → **é** [0,10] | **sinto** [0,10] | **cheira** [0,10] **| fede** [0,05] |. . .
Adjetivo → **direita** [0,10] | **morto** [0,05] | **fedorento** [0,02] | **com brisa** [0,02]. . .
Advérbio → **aqui** [0,05] | **a frente** [0,05] | **próximo** [0,02] |. . .
Pronome → **me** [0,10] | **você** [0,03] | **eu** [0.10] | **ele** [0,10] |. . .
ProRel → **que** [0,40] | **o qual** [0,15] **| quem** [0,20] | **quem** [0,02] ∨... *Nome* → **João**

[0,01] | **Maria** [0,01] | **Boston** [0,01] |. . .
Artigo → **o** [0,40] | **um** [0,30] | **um** [0,10] | **todo** [0,05] |. . .
Prep → **para** [0,20] | **em** [0,10] | **sobre** [0,05] | **perto** [0,10] |. . .
Conj → **e** [0,50] | **ou** [0,10] | **mas** [0,20] | **ainda** [0,02] ∨... *Dígito* → **0** [0,20] | **1** [0,20] |
2 [0,20] | **3** [0,20] | **4** [0,20] |. . .

Figura 23.1 O léxico para ε_0. *ProRel* é abreviatura de pronome relativo, *Prep* de preposição e *Conj* de conjunção. A soma das probabilidades para cada categoria é 1.

Cada uma das categorias termina em… para indicar que existem outras palavras na categoria. Porém, devemos observar que existem duas razões distintas para a ausência de palavras. No caso de substantivos, verbos, adjetivos e advérbios, em princípio é impossível listar todos eles. Não apenas existem dezenas ou milhares de membros em cada classe, mas novos nomes — como *iPOD* ou *biodiesel* — estão sendo acrescentados constantemente. Essas cinco categorias são chamadas **classes abertas**. Para as categorias de pronome, pronome relativo, artigo, preposição e conjunção poderíamos ter listado todas as palavras com um pouco mais de trabalho. Essas são chamadas **classes fechadas**. Elas têm um pequeno número de palavras (de algumas unidades a algumas dezenas) que podem, em princípio, ser totalmente enumeradas. As classes fechadas mudam no decorrer de séculos, não de meses. Por exemplo, "thee" e "thou" eram pronomes de uso comum no século XVII, estavam em declínio no século XIX e hoje só são vistos em poesia e em alguns dialetos regionais.

23.1.2 A gramática de ε_0

A próxima etapa é combinar as palavras para formar sintagmas. A Figura 23.2 mostra uma gramática para ε_0, com regras para cada uma das seis categorias sintáticas e um exemplo para cada regra reescrita.[2] A Figura 23.3 mostra uma **árvore de análise sintática** para a sentença "Todos os wumpus cheiram". A árvore de análise sintática dá uma prova construtiva de que a cadeia de palavras é realmente uma sentença de acordo com as regras de ε_0. A gramática ε_0 gera grande variedade de sentenças em inglês como as seguintes:

$\mathcal{E}_0$: *S*	→	*SN SV*	[0,90] Eu + sinto uma brisa
	|	S Conj S	[0,10] Eu sinto uma brisa + e + ela fede
SN	→	*Pronome*	[0,30] Eu
	|	*Nome*	[0,10] João
	|	*Substantivo*	[0,10] poços
	|	*Artigo Substantivo*	[0,25] o + wumpus
	|	*Artigo Adjs Substantivo*	[0,05] o + fede a morto + wumpus
	|	*Dígito Dígito*	[0,05] 3 4
	|	*SN SP*	[0,10] o wumpus + em 1 3
	|	*SN ClausRel*	[0,05] o wumpus + que é fedido
SV	→	*Verbo*	[0,40] fede
	|	*SV SN*	[0,35] sinto + uma brisa
	|	*SV Adjetivo*	[0,05] cheira a + morto
	|	*SV SP*	[0,10] está + em 1 3
	|	*SV Advérbio*	[0,10] vai + para a frente
Adjs	→	*Adjetivo*	[0,80] fedorento
	|	*Adjetivo Adjs*	[0,20] fedorento + morto
SP	→	*Prep SN*	[1,00] para + o leste
ClauRel	→	*ProRel SV*	[1,00] isso + é fedorento

Figura 23.2 A gramática para ε_0, com frases de exemplo para cada regra. As categorias sintáticas são sentenças (*S*), sintagma nominal (*SN*), sintagma verbal (*SV*), lista de adjetivos (*Adjs*), sintagma preposicional (*SP*) e cláusula relativa (*ClauRel*).

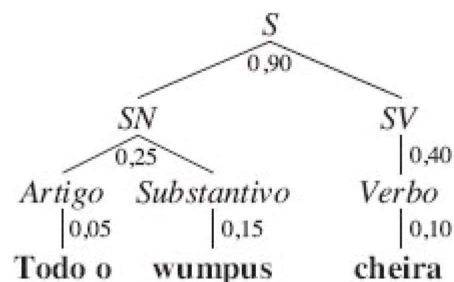


Figura 23.3 Árvore de análise sintática para a sentença "Todo wumpus cheira", de acordo com a gramática ε_0. Cada nó interior da árvore é rotulado com sua probabilidade. A probabilidade da árvore como um todo é de $0{,}9 \times 0{,}25 \times 0{,}05 \times 0{,}15 \times 0{,}40 \times 0{,}10 = 0{,}0000675$. Uma vez que essa árvore é a única análise sintática da sentença, esse número também é a probabilidade da sentença. A árvore também pode ser escrita na forma linear como [*S* [*SN* [*Artigo* **todos os**] [*Substantivo* **wumpus**]] [*SV* [*Verbo* **cheiram**]]].

João está no poço
O wumpus que fede está no 2 2
Maria está em Boston e o wumpus está perto do 3 2.

Infelizmente, a gramática **superproduz**, isto é, gera sentenças que não são gramaticais, como "Mim ir Boston" e "Eu cheiro poços wumpus João". Também **subproduz**: existem muitas sentenças em inglês que ela rejeita, como "Eu acho que o wumpus é fedorento". Veremos como aprender uma gramática melhor mais tarde, por ora nos concentraremos no que podemos fazer com a gramática que temos.

23.2 ANÁLISE SINTÁTICA

A **análise sintática** é o processo de analisar uma cadeia de palavras para descobrir a sua estrutura frasal, de acordo com as regras de uma gramática. A Figura 23.4 mostra que podemos começar com o símbolo *S* e pesquisar de cima para baixo em uma árvore que tem as palavras como suas folhas ou podemos começar com as palavras e buscar de baixo para cima de uma árvore que culmina em um *S*. Contudo, tanto a análise de cima para baixo como a de baixo para cima podem ser ineficientes porque podem acabar repetindo esforços em áreas de espaço de busca que levam a becos sem saída. Considere as duas sentenças a seguir:

Lista de itens	*Regra*
S	
SN SV	*S* → *NP SV*
SN SV Adjetivo	*SV* → *SV Adjetivo*
SN Verbo Adjetivo	*SV* → *Verbo*
SN Verbo **morto**	*Adjetivo* → **morto**
SN **está morto**	*verbo* → **está**
Artigo Substantivo **está morto**	*SN* → *Artigo Substantivo*
Artigo **wumpus está morto**	*Substantivo* → **wumpus**
o wumpus está morto	*Artigo* → **o**

Figura 23.4 Traçado do processo para encontrar uma análise para a cadeia "O wumpus está morto" como sentença, de acordo com a gramática ε_0. Visto como análise de cima para baixo, começamos com a lista de itens *S* e, em cada etapa, correspondem a um item *X* com uma regra da forma ($X \rightarrow \ldots$) e substituímos *X* na lista de itens com (…). Visto como análise de baixo para cima, começamos com a lista de itens que são as palavras da sentença e, em cada etapa, correspondem a uma sequência de tokens (…) na lista contra uma regra da forma ($X \rightarrow \ldots$) e substituímos (…) por *X*.

Have the students in section 2 of Computer Science 101 take the exam.
Have the students in section 2 of Computer Science 101 taken the exam?

Embora compartilhem as 10 primeiras palavras, essas sentenças têm análises sintáticas muito diferentes porque a primeira é um comando e a segunda é uma pergunta. Um algoritmo de análise da esquerda para a direita teria de pressupor que a primeira palavra é parte de um comando ou de uma pergunta, e não seria capaz de saber se a suposição é correta até pelo menos a décima primeira palavra, *take* ou *taken*. Se fizer a suposição errada, o algoritmo terá de percorrer de volta toda a distância até a primeira palavra e analisar novamente toda a sentença sob outra interpretação.

Para evitar essa fonte de ineficiência, podemos usar programação dinâmica: *cada vez que analisamos uma subcadeia, armazenamos os resultados de modo que não tenhamos que reanalizá-los mais tarde*. Por exemplo, uma vez que descobrimos que "os alunos na seção 2 de Ciência da Computação 101" é um *SN*, podemos registrar o resultado em uma estrutura de dados conhecida

como **diagrama**. Os algoritmos que fazem isso são chamados de **analisadores de diagrama**. Por estarmos tratando com gramática livre de contexto, qualquer sintagma que for encontrado no contexto de uma ramificação do espaço de busca pode funcionar muito bem em qualquer outra ramificação de espaço de busca. Existem muitos tipos de analisadores de diagrama; descreveremos uma versão de baixo para cima chamada de **algoritmo CYK**, que tem por trás seus inventores, John Cocke, Daniel Younger e Tadeo Kasami.

O algoritmo CYK é mostrado na Figura 23.5. Note que ele exige uma gramática com todas as regras em um de dois formatos muito específicos: regras lexicais da forma $X \rightarrow$ **palavra** e regras sintáticas da forma $X \rightarrow Y\,Z$. Esse formato da gramática, chamado de **forma normal de Chomsky**, pode parecer restritivo mas não é: qualquer gramática livre de contexto pode ser transformada automaticamente na forma normal de Chomsky. O Exercício 23.8 conduz pelo processo.

função ANÁLISE-CYK (palavras, gramática) **retorna** *P*, uma tabela de probabilidades
  $N \leftarrow$ COMPRIMENTO (*palavras*)
  $M \leftarrow$ o número de símbolos não terminais na *gramática*
  $P \leftarrow$ uma matriz de tamanho $[M, N, N]$, inicialmente todos 0
  / * *Inserir regras lexicais para cada palavra* * /
  para $i = 1$ **até** N **faça**
    para cada regra da forma ($X \rightarrow palavras_i\ [p]$) **faça**
      $P[X, i, 1] \leftarrow p$
  / * *Combine a primeira e a segunda partes do lado direito das regras, da curta para a longa* * /
  para comprimento = 2 **até** N **faça**
    para início = 1 **até** $N -$ comprimento + 1 **faça**
      para *comp1* = 1 **até** $N - 1$ **faça**
        comp2 $\leftarrow$ comprimento − *comp1*
        para cada regra da forma ($X \rightarrow Y\,Z\ [p]$) **faça**
          $P[X, início, comprimento] \leftarrow$ MAX($P[X, início, comprimento]$,
          $P[Y, início, comp1] \times P[Z, início + comp1, comp2] \times p$)
  retornar *P*

Figura 23.5 O algoritmo CYK para análise sintática. Dada uma sequência de palavras, ele encontra a derivação mais provável para a sequência inteira e para cada subsequência. Retorna a tabela inteira, *P*, em que uma entrada *P*[*X*, *início*, *comp*] é a probabilidade do X mais provável de comprimento *comp* iniciando na posição *início*. Se não houver um *X* desse tamanho no local, a probabilidade será 0.

O algoritmo CYK utiliza o espaço de $O(n^2m)$ da tabela *P*, onde *n* é o número de palavras na sentença e *m* é o número de símbolos não terminais na gramática, que leva o tempo $O(n^3m)$. (Uma vez que *m* é constante para uma gramática particular, é comumente descrito como $O(n^3)$.) Nenhum algoritmo pode fazer melhor para gramáticas livres de contexto em geral, embora haja algoritmos mais rápidos em gramáticas mais restritas. Na verdade, é quase um truque para que o algoritmo

complete no tempo $O(n^3)$, dado que é possível que uma sentença tenha um número exponencial de árvores de análise sintática. Considere a sentence

Fall leaves fall and spring leaves spring.

É ambíguo porque cada palavra (exceto "and") pode ser um substantivo ou um verbo, e "fall"e "spring" podem ser também adjetivos. (Por exemplo, um significado de "Fall leaves fall" é equivalente a "Outono abandona outono.) Com ε_0, a sentença tem quatro análises:

[*S* [*S* [*SN* Fall leaves] fall] e [*S* [*SN* spring leaves]spring]]
[*S* [*S* [*SN* Fall leaves] fall] e [*S* spring [*SV* leaves spring]]
[*S* [*S* Fall [*SV* leaves fall]] e [*S* [*SN* spring leaves] spring]
[*S* [*S* Fall [*SV* leaves fall]] e [*S* spring [*SV* leaves spring]].

Se tivéssemos subsentenças conjuntas *c* ambíguas de duas maneiras, teríamos 2^c formas de escolher análises para as subsentenças.[3] Como o algoritmo CYK processa essas árvores de análise sintática de 2^c no tempo $O(c^3)$? A resposta é que ele não examina todas as árvores de análise; tudo o que faz é o cálculo da probabilidade da árvore mais provável. As subárvores estão todas representadas na tabela *P* e, com um pouco de trabalho, poderíamos enumerar todas (em tempo exponencial), mas a beleza do algoritmo CYK é que não temos que enumerá-las, a menos que queiramos.

Na prática, geralmente não estamos interessados em todas as análises, apenas nas melhores ou nas poucas melhores. Pense no algoritmo CYK como a definição do espaço de estados completo definido pelo operador "aplicar regra de gramática". É possível pesquisar apenas uma parte desse espaço usando a busca A*. Cada estado nesse espaço é uma lista de itens (palavras ou categorias), como mostrado na tabela analítica da base para o topo (Figura 23.4). O estado inicial é uma lista de palavras, e um estado objetivo é o único item *S*. O custo de um estado é o inverso de sua probabilidade, tal como definido pelas regras aplicadas até agora, e há várias heurísticas para estimar a distância que falta para a meta, a melhor heurística vindo de aprendizagem de máquina aplicada a um *corpus* de sentenças. Com o algoritmo A* não temos de buscar o espaço de estado inteiro, e temos a garantia de que a primeira análise encontrada será a mais provável.

23.2.1 Aprendizagem de probabilidades para GLCPs

Uma GLCP tem muitas regras, com uma probabilidade para cada regra. Isso sugere que a **aprendizagem** da gramática a partir de dados pode ser melhor do que uma abordagem de engenharia do conhecimento. A aprendizagem será mais fácil se nos for dado um *corpus* de sentenças analisadas corretamente, comumente chamado de **floresta sintática**. O Penn Treebank (Marcus *et al*., 1993) é o mais conhecido, consistindo em três milhões de palavras que foram anotadas com parte da fala e estrutura da árvore de análise sintática, usando o trabalho humano assistido por algumas ferramentas automatizadas. A Figura 23.6 mostra uma árvore anotada do Penn Treebank.

```
[[S [SN-SUJ-2 Her eyes]
   [SV were
      [SV glazed
         [SN *-2]
         [SBAR-ADV, as if
            [S [SN-SUJ she]
               [SV didn't
                  [SV [SV hear [SN *-1]]
                     ou
                     [SV [ADSV even] see [SN *-1]]
                     [SN-1 him]]]]]]]]
   .]
```

Figura 23.6 Árvore anotada para a sentença "Her eyes were glazed as if she didn't hear or even see him" do Penn Treebank. Observe que, nessa gramática, há distinção entre um sintagma nominal objeto (*SN*) e um sintagma nominal sujeito (*SN-SUJ*). Observe também um fenômeno gramatical que ainda não abrangemos: o movimento de sintagma de uma parte da árvore para a outra. Essa árvore analisa a frase "hear or even see him" como consistindo em dois constituintes de *SV*s, [SV hear [SN *-1]] e [SV [ADSV even] see [SN *-1]], sendo que ambos têm um objeto faltante, denotado como *-1, que se refere ao *SN* rotulado em outras partes da árvore como [SN-1 him].

Dado um *corpus* de árvores, podemos criar um GLCP apenas pela contagem (e alisamento). No exemplo anterior, existem dois nós da forma [*S*[*SN*…] [*SV*…]]. Contaríamos estes, e todas as outras subárvores com raiz *S* no *corpus*. Se existirem 100.000 nós *S* dos quais 60.000 são dessa forma, criaremos a regra:

$$S \rightarrow SN\ SV\ [0,60].$$

E, se uma floresta sintática não estiver disponível mas tivermos um *corpus* de de sentenças brutas sem rótulo? Ainda é possível aprender uma gramática de tal *corpus*, mas é mais difícil. Primeiro, realmente temos dois problemas: aprender a estrutura das regras de gramática e aprender as probabilidades associadas a cada regra (tivemos a mesma distinção no aprendizado das redes de Bayes). Vamos supor que nos foram dados os nomes das categorias lexicais e sintáticas (se não, podemos simplesmente assumir as categorias, X_1….. X_n e o uso de validação cruzada para escolher o melhor valor de *n*). Podemos então assumir que a gramática inclui todas as regras possíveis ($X \rightarrow Y$ Z) ou ($X \rightarrow palavra$), embora muitas dessas regras tivessem probabilidade 0 ou próximo de 0.

Podemos então usar a abordagem da maximização de expectativa (EM), assim como fizemos no aprendizado de MOMs. Os parâmetros que estamos tentando aprender são as probabilidades da regra; iniciaremos com valores aleatórios ou uniformes. As variáveis ocultas são as árvores de análise: não sabemos se uma sequência de palavras w_i… w_j é ou não gerada por uma regra ($X \rightarrow$ …). A etapa E estima a probabilidade de que cada subsequência seja gerada por cada regra. A etapa M, então, estima a probabilidade de cada regra. O cálculo todo pode ser feito em programação dinâmica, com um algoritmo chamado de **algoritmo dentro-fora**, em analogia ao algoritmo para a

frente e para trás de MOMs.

O algoritmo dentro-fora parece mágico na medida em que influencia uma gramática de um texto não analisado. Mas tem vários inconvenientes. Primeiro, as análises que são atribuídas pelas gramáticas influenciadas são muitas vezes difíceis de entender e insatisfatórias para os linguistas. Isso torna difícil combinar conhecimentos artesanais com a indução automática. Segundo, é lento: $O(n^3m^3)$, onde n é o número de palavras em uma frase e m é o número de categorias gramaticais. Terceiro, o espaço de atribuições de probabilidades é muito grande e, empiricamente, parece que ficar preso em máximas locais é um problema grave. Alternativas como têmpora simulada podem se aproximar do máximo global, a um custo de ainda mais computação. Lari e Young (1990) concluíram que o algoritmo dentro-fora é "intratável computacionalmente para problemas reais".

No entanto, pode ser feito progresso se estivermos dispostos a pisar fora dos limites da aprendizagem exclusivamente do texto não analisado. Uma abordagem é aprender com **protótipos**: semear o processo uma dúzia ou duas de regras, similares às regras em ε_1. Daí, regras mais complexas podem ser aprendidas mais facilmente, e a gramática resultante realiza a análise do inglês com revocação e precisão geral das sentenças de cerca de 80% (Haghighi e Klein, 2006). Outra abordagem é usar florestas sintáticas, mas, além de aprender diretamente as regras GLPC a partir do agrupamento com colchetes, aprende-se também distinções que não estão na floresta sintática. Por exemplo, não significa que a árvore na Figura 23.6 faça distinção entre SN e $SN - SUJ$. O último é utilizado para o pronome "she", o anterior para o pronome "her". Vamos explorar essa questão na Seção 23.6; por enquanto vamos apenas dizer que há muitas maneiras como seria útil **dividir** uma categoria como SN — um sistema de gramática de indução que usa florestas sintáticas, mas dividir categorias automaticamente é melhor do que aqueles que preservam o conjunto da categoria original (Petrov e Klein, 2007c). As taxas de erro para gramáticas aprendidas automaticamente ainda são cerca de 50% maiores do que as gramáticas construídas manualmente, mas a diferença está diminuindo.

23.2.2 Comparação entre modelos livres de contexto e modelos de Markov

O problema com GLCPs é que eles são livres de contexto. Isso significa que a diferença entre P("comer uma banana") e P("comer uma bandana") depende apenas de P(substantivo → "banana") *versus* P(substantivo → "bandana") e não da relação entre "comer" e os respectivos objetos. Um modelo de Markov de ordem dois ou mais, dado um *corpus* suficientemente grande, vai saber que "comer uma banana" é mais provável. Podemos combinar um modelo GLCP e de Markov para obter o melhor de ambos. A abordagem mais simples é estimar a probabilidade de uma sentença com a média geométrica das probabilidades calculadas por ambos os modelos. Então saberíamos que "comer uma banana" é provável, tanto do ponto de vista gramatical como lexical. Mas ainda não iria extrair a relação entre "comer" e "banana" em "comer uma banana ligeiramente envelhecida, mas ainda palatável" porque aqui a relação é de mais do que duas palavras de distância. O aumento da ordem do modelo de Markov não vai obter precisamente a relação, para fazer isso podemos utilizar um GLCP lexicalizado, como descrito na próxima seção.

Outro problema com GLCPs é que eles tendem a ter uma preferência muito forte para sentenças

mais curtas. Em um *corpus* tal como o *Wall Street Journal*, a duração média de uma sentença é de cerca de 25 palavras. Mas um GLCP geralmente vai atribuir uma probabilidade bastante alta para muitas sentenças curtas, como “Ele dormiu”, enquanto no *Journal* estamos mais propensos a ver algo como “Foi relatado por uma fonte confiável que a alegação de que ele dormiu é crível”. Parece que as frases no *Journal* realmente não são livres de contexto; em vez disso, os escritores têm uma ideia do comprimento de sentença esperado e usam esse comprimento como uma restrição global suave sobre suas sentenças. É difícil refletir isso em um GLCP.

23.3 GRAMÁTICAS AUMENTADAS E INTERPRETAÇÃO SEMÂNTICA

Nesta seção vamos ver como estender gramáticas livres de contexto — para dizer que, por exemplo, nem todo *SN* é independente de contexto, mas certos *SN*s são mais propensos a aparecer em um contexto e outros em outro contexto.

23.3.1 GLCPs lexicalizados

Para chegar na relação entre o verbo “comer” e os substantivos “banana” *versus* “bandana”, podemos usar um **GLCP lexicalizado**, em que as probabilidades de uma regra dependem da relação entre as palavras na árvore de análise, não apenas sobre a adjacência de palavras em uma sentença. Certamente, não podemos fazer com que a probabilidade dependa de cada palavra na árvore porque não vamos ter dados de treinamento suficientes para estimar todas as probabilidades. É útil para introduzir a noção da **cabeça** de uma sintagma — a palavra mais importante. Assim, “comer” é a cabeça do *SV* “comer uma banana” e “banana” é a cabeça do *SN* “uma banana”. Usamos a notação *SV*(*v*) para denotar uma frase com a categoria *SV* cuja palavra cabeça é *v*. Dizemos que a categoria *SV* está **aumentada** com a variável cabeça *v*. Apresentamos uma **gramática aumentada** que descreve a relação verbo-objeto:

$SV(v) \rightarrow Verbo(v)\ SN(n)$	$[P_1(v, n)]$
$SV(v) \rightarrow Verbo(v)$	$[P_2(v)]$
$SN(n) \rightarrow Artigo(a)\ Adjs(j)\ Substantivo(n)$	$[P_3(n, a)]$
Substantivo (**banana**) → **banana**	$[p_n]$
...	...

Aqui, a probabilidade $P_1(v, n)$ depende das palavras cabeça *v e n*. Definiremos essa probabilidade como relativamente alta quando *v* for “comer” e *n* for “banana”, e baixa quando *n* for “bandana”. Observe que, desde que estamos considerando apenas os cabeças, a distinção entre “comer uma banana” e “comer uma banana podre” não será pega pelas probabilidades. Outro problema com essa abordagem é que, em um vocabulário com, digamos, 20.000 substantivos e 5.000 verbos, P_1 necessita de 100 milhões de estimativas de probabilidade. Apenas uma pequena porcentagem dela pode vir de um *corpus*, o resto terá de vir do alisamento (veja a Seção 22.1.2). Por exemplo, podemos estimar $P_1(v, n)$ para um par (*v, n*) que muitas vezes não vimos (ou em definitivo) recuando a um modelo que depende somente de *v*. Essas probabilidades sem propósito ainda são

muito úteis, pois podem capturar a distinção entre um verbo transitivo, como "comer"— que terá valor alto para P_1 e valor baixo para P_2 — e um verbo intransitivo como "dormir", que terá o sentido inverso. É bastante viável aprender essas probabilidades de uma floresta sintática.

23.3.2 Definição formal de regras de gramática aumentada

As regras aumentadas são complicadas, por isso vamos dar uma definição formal, mostrando como uma regra aumentada pode ser traduzida em uma sentença lógica. A sentença terá a forma de cláusula definida, de modo que o resultado é chamado de **gramática de cláusula definida** ou GCD. Usaremos como exemplo uma versão de uma regra de gramática lexicalizada de *SN* com uma nova peça de notação:

SN(n) → Artigo(a) Adjs(j) Substantivo(n) {Compatível(j, n)}.

A novidade é a notação {*restrição*} para denotar uma restrição lógica em algumas das variáveis; a regra só vale quando a restrição é verdadeira. Aqui, o predicado *Compatível*(*j, n*) serve para testar se o adjetivo *j* e o substantivo *n* são compatíveis; eles seriam definidos por uma série de afirmações como *Compatível*(**preto**, **cão**). Podemos converter essa regra gramatical em uma cláusula definida (1) invertendo a ordem dos lados direito e esquerdo (2), fazendo uma conjunção de todos os componentes e restrições (3), adicionando uma variável para *si* à lista de argumentos para cada constituinte, para representar a sequência de palavras abrangidas pelo componente, (4) adicionando um termo para a concatenação de palavras, *Juntar*(s_1,…) à lista de argumentos da raiz da árvore. Isso nos dá

$$Artigo(a, s1) \wedge Adjs(j, s_2) \wedge Substantivo(n, s3) \wedge Compatível(j, n)$$
$$\Rightarrow SN(n, Juntar\ (s_1, s_2, s_3)).$$

Essa cláusula definida diz que, se o predicado *artigo* for verdadeiro de uma palavra cabeça *a* e uma sequência s_1, e *Adjs* igualmente for verdadeiro de uma palavra cabeça *j* e uma sequência s_2, e *substantivo* for verdadeiro de uma palavra cabeça *n* e uma sequência s_3, e se *j* e *n* forem compatíveis, então o predicado *NP* é verdadeiro da palavra cabeça *n* e do resultado de juntar as cadeias s_1, s_2 *e* s_3.

A tradução da GCD deixou de fora as probabilidades, mas poderíamos colocá-las de volta: basta aumentar cada componente com mais uma variável que represente a probabilidade do componente e aumentar a raiz com uma variável que seja o produto das probabilidades componentes vezes a probabilidade da regra.

A tradução da regra gramatical para cláusula definida nos permite falar sobre a análise como inferência lógica. Isso torna possível raciocinar sobre linguagens e cadeias de muitas maneiras diferentes. Por exemplo, significa que podemos fazer análise do topo para a base utilizando encadeamento para a frente ou análise do topo para a base utilizando encadeamento para trás. Na verdade, a análise da linguagem natural com GCDs foi uma das primeiras aplicações da linguagem lógica de programação Prolog (e a motivação para ela). Às vezes, é possível executar o processo

para trás e fazer a **geração da linguagem**, bem como da análise. Por exemplo, seguindo para a Figura 23.10, pode ser dada a forma semântica *Loves*(*John, Mary*) em um programa lógico e aplicar as regras de cláusula definida para deduzir

S(*Loves*(*John, Mary*), [***John, loves, Mary***]).

Isso funciona para exemplos pequenos, mas para sistemas de geração de linguagem séria é preciso mais controle sobre o processo do que é proporcionado sozinho pelas regras de GCD.

23.3.3 Concordância de caso e de sujeito e verbo

Vimos na Seção 23.1 que a gramática simples para ε_0 superproduz, gerando sentenças como "Me sentir um mau cheiro". Para evitar esse problema, nossa gramática teria de saber que "me" não é um *SN* válido quando for o assunto de uma sentença. Os linguistas dizem que o pronome "eu" está no caso subjetivo e "me" está no caso acusativo.[4] Podemos explicar isso dividindo *SN* em duas categorias, SN_S e SN_O, para representar sintagmas nominais no caso subjetivo e objetivo, respectivamente. Também precisamos dividir a categoria *Pronome* em duas categorias $Pronome_S$ (que inclui "eu") e $Pronome_O$ (que inclui "me"). A parte superior da Figura 23.7 mostra a gramática para a **concordância de caso**; chamamos a linguagem resultante de ε_1. Observe que todas as regras *SN* devem ser duplicadas, uma vez para SN_S e uma vez para SN_O.

$\mathcal{E}_1$:	S	→	$SN_S\ SV$ | . . .
	SN_S	→	$Pronome_S$ | *Nome* | *Substantivo* | . . .
	SN_O	→	$Pronome_O$ | *Nome* | *Substantivo* | . . .
	SV	→	$SV\ SN_O$ | . . .
	SP	→	$Prep\ SN_O$
	$Pronome_S$	→	**eu** | **você** | **ele** | **ela** | . . .
	$Pronome_O$	→	**meu** | **teu** | **seu** | **sua** | . . .
			. . .
$\mathcal{E}_2$:	*S (cabeça)*	→	*SN (Suj, pn, h) SV (pn, cabeça)* | . . .
	SN (c, pn, cabeça)	→	*Pronome (c, pn, cabeça)* | *Substantivo (c, pn, cabeça)* | . .
	SV (pn, cabeça)	→	*SV (pn, cabeça) SN (Obj, p, h)* | . . .
	SP (cabeça)	→	*Prep (cabeça) SN (Obj, pn, h)*
	Pronome (Suj, 1S, **Eu***)*	→	**Eu**
	Pronome (Suj, 1P, **nós***)*	→	**nós**
	Pronome (Obj,1S, **me***)*	→	**me**
	Pronome (Obj, 3P, **eles***)*	→	**eles, elas / os, as**
			. . .

Figura 23.7 Parte superior: parte de uma gramática para a linguagem ε_1, que lida com casos subjetivos e objetivos em sintagmas nominais e, portanto, não superproduz tanto como ε_0. As partes que são idênticas a ε_0 foram omitidas. Parte inferior: parte de uma gramática aumentada para ε_2, com três aumentos: concordância de caso, concordância de sujeito e verbo, e palavra cabeça. *Suj*, *Obj*, *1S*, *1P* e *3P* são constantes, e os nomes em minúsculas são variáveis.

Infelizmente, ε_1 ainda superproduz. O inglês requer a **concordância de sujeito e verbo** para a pessoa e de número do sujeito e do verbo principal de uma sentença. Por exemplo, se "eu" for o sujeito, então "eu cheiro" é gramatical, mas "eu cheira" não é. Se "ele" for o sujeito, temos o inverso. Em inglês, as distinções de concordância são mínimas: a maioria dos verbos tem uma forma para o sujeito da terceira pessoa do singular (*he*, *she* ou *it*) e uma segunda forma para todas as outras combinações de pessoa e número. Há uma exceção: o verbo "to be" tem três formas: "*I am* / *you are* / *he is*". Então, uma distinção (caso) divide o *SN* de duas maneiras, outra distinção (pessoa e número) divide o *SN* de três maneiras, e, ao descobrirmos outras distinções terminaríamos com um número exponencial de formas de *SN* subscritas se tomássemos a abordagem de ε_1. Os aumentos são uma abordagem melhor: podem representar um número exponencial de formas como uma regra única.

Na parte inferior da Figura 23.7 vemos (parte de) uma gramática aumentada para o idioma ε_2, que trata da concordância de caso, concordância de sujeito-verbo e das palavras cabeça. Temos apenas uma categoria de *SN*, mas o *SN* (*c*, *pn*, *cabeça*) tem três aumentos: *c* é um parâmetro do caso, *pn* é um parâmetro de pessoa e número, e *cabeça* é um parâmetro para a palavra cabeça do sintagma. As outras categorias também são aumentadas com cabeças e outros argumentos. Vamos considerar uma regra em detalhe:

S(cabeça) → SN(Suj, pn, h) SV(pn, cabeça).

Essa regra é mais fácil de entender da direita para a esquerda: quando um *SN* e um *SV* são unidos, eles formam um *S*, mas apenas se *SN* estiver no caso subjetivo (*Sbj*) e a pessoa e o número (*pn*) do *SN* e do *SV* forem idênticas. Se isso for válido, temos um *S* cuja cabeça é a mesma que a do *SV*. Observe que a cabeça do *SN*, denotada pela variável fictícia *h*, não faz parte do aumento de *S*. As regras lexicais para ε_2 preenchem os valores dos parâmetros e também são mais bem entendidas lendo-se da direita para a esquerda. Por exemplo, a regra

Pronome(*Suj*, *1S,* **eu**) → **eu**

diz que "eu" pode ser interpretado como um *pronome* no caso subjetivo, primeira pessoa do singular, com a cabeça "eu". Para simplificar, omitimos as probabilidades dessas regras, mas o aumento funciona com probabilidades. O aumento também pode funcionar com mecanismos de aprendizagem automática. Petrov e Klein (2007c) mostraram como um algoritmo de aprendizagem pode dividir automaticamente a categoria *SN* em SN_S e SN_O.

23.3.4 Interpretação semântica

Para mostrar como adicionar semântica a uma gramática, começamos com um exemplo que é mais simples que o inglês: a semântica de expressões aritméticas. A Figura 23.8 mostra uma gramática para expressões aritméticas, em que cada regra é aumentada com uma variável que indica a interpretação semântica do sintagma. A semântica de um dígito como "3" é o dígito em si. A semântica de uma expressão como "3 + 4" é o operador "+" aplicado à semântica do sintagma "3" e

"4". As regras obedecem ao princípio da **semântica composicional** — a semântica de um sintagma é função da semântica dos subsintagmas. A Figura 23.9 mostra a árvore de análise para 3 + (4 ÷ 2) de acordo com essa gramática. A raiz da árvore de análise é *Exp*(5), uma expressão cuja interpretação semântica é 5.

$$Exp(x) \rightarrow Exp(x_1)\ Operador(op)\ Exp(x_2)\ \{x = Aplicar(op, x_1, x_2)\}$$
$$Exp(x) \rightarrow (\ Exp(x)\ )$$
$$Exp(x) \rightarrow Número(x)$$
$$Número(x) \rightarrow Dígito(x)$$
$$Número(x) \rightarrow Número(x_1)\ Dígito(x_2)\ \{x = 10 \times x_1 + x_2\}$$
$$Dígito(x) \rightarrow x\ \{0 \leq x \leq 9\}$$
$$Operador(x) \rightarrow x\ \{x \in \{+, -, \div, \}\}$$

Figura 23.8 Uma gramática para expressões aritméticas, aumentada com semântica. Cada variável *xi* representa a semântica de um componente. Observe o uso da notação {*teste*} para definir predicados lógicos que devem ser satisfeitos, mas que não são componentes.

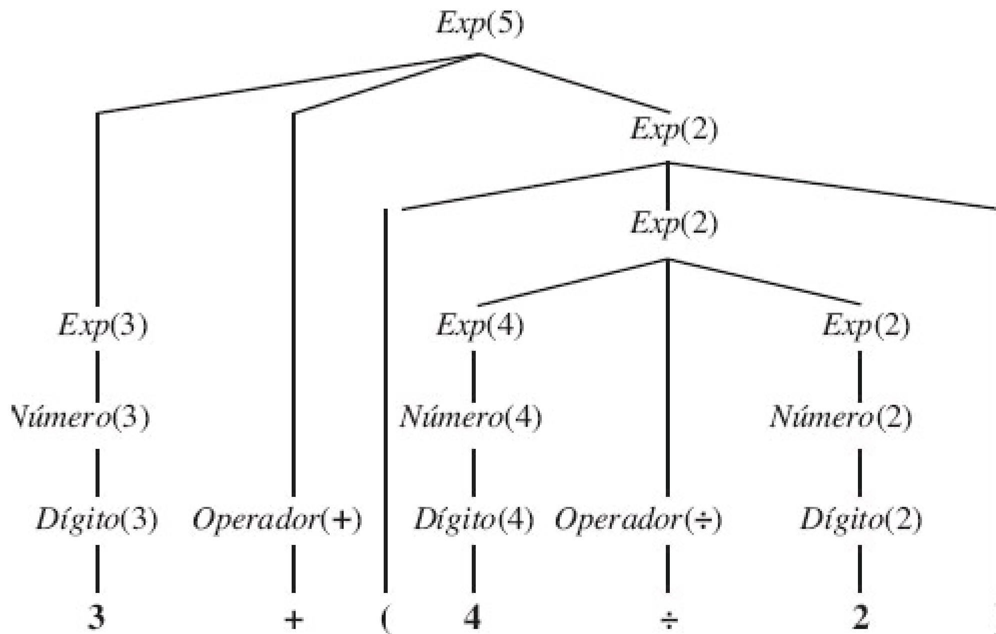


Figura 23.9 Árvore de análise com interpretações semânticas para a cadeia "3 + (4 ÷ 2)".

Agora vamos passar à semântica do inglês ou, pelo menos, de ε_0. Vamos começar determinando quais representações semânticas queremos associar a cada um dos sintagmas. Usaremos a sentença de exemplo simples "John loves Mary" ("John ama Mary"). O *SN* "John" deve ter como sua interpretação semântica o termo lógico *John*, e a sentença como um todo deve ter como sua interpretação a sentença lógica *Loves*(*John*, *Mary*). Isso parece muito claro. A parte complicada é o *SV* "loves Mary". A interpretação semântica desse sintagma não é nem um termo lógico nem uma sentença lógica completa. Intuitivamente, "loves Mary" é uma descrição que poderia ou não se aplicar a uma pessoa em particular (nesse caso, ela se aplica a John). Isso significa que "loves Mary" é um **predicado** que, quando combinado com um termo que representa uma pessoa (a pessoa que ama), produz uma sentença lógica completa. Usando a notação λ, podemos representar "loves Mary" como o predicado

$$\lambda x\ Loves(x, Mary).$$

Agora, precisamos de uma regra que informe que "um *SN* com semântica *obj* seguido por um *SV* com semântica *rel* gera uma sentença cuja semântica é o resultado de se aplicar *rel* a *obj"*:

$$S(rel(obj)) \rightarrow SN(obj)\ SV(rel).$$

A regra nos diz que a interpretação semântica de "John loves Mary" é

$$(\lambda x\ Loves(x, Mary))(John),$$

que é equivalente a $Loves(John, Mary)$.

O restante da semântica decorre de modo direto das escolhas que fizemos até agora. Como *SV*s são representados como predicados, é uma boa ideia manter a consistência e representar verbos também como predicados. O verbo "loves" é representado como $\lambda y\ \lambda x\ Loves(x, y)$, o predicado que, ao receber o argumento *Mary*, retorna o predicado $\lambda x\ Loves(x, Mary)$. Finalizamos com a gramática mostrada na Figura 23.10 e com a árvore de análise mostrada na Figura 23.11. Poderíamos facilmente ter adicionado semântica para ε_2; optamos por trabalhar com ε_0 para que o leitor possa se concentrar em um tipo de aumento de cada vez.

$S(rel(obj)) \rightarrow SN(obj)\ SV(rel)$
$SV(rel(obj)) \rightarrow Verbo(rel)\ SN(obj)$
$SN(obj) \rightarrow Nome(obj)$

$Nome(John) \rightarrow$ **John**
$Nome(Mary) \rightarrow$ **Mary**
$Verbo\ (\lambda x\ \lambda y\ Loves\ (x, y)) \rightarrow$ **loves**

Figura 23.10 Uma gramática que pode derivar uma árvore de análise e uma interpretação semântica para "John loves Mary" (e três outras sentenças). Cada categoria é aumentada com um único argumento representando a semântica.

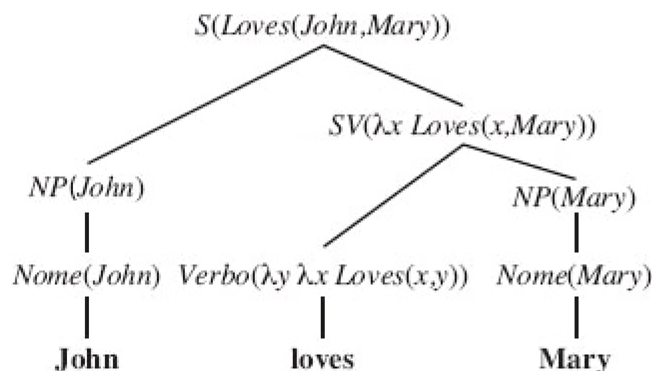


Figura 23.11 Uma árvore de análise com interpretações semânticas para a cadeia "John loves Mary".

Adicionar aumentos semânticos para uma gramática manualmente é trabalhoso e propenso a erros.

Portanto, tem havido vários projetos para aprender aumentos semânticos a partir de exemplos. CHILL (Zelle e Mooney, 1996) é uma lógica de programação indutiva (LPI), que aprende gramática e um analisador especializado para aquela gramática a partir de exemplos. O domínio de destino são as consultas de banco de dados de linguagem natural. Os exemplos de treinamento consistem em pares de sequências de palavras e das formas semânticas correspondentes, por exemplo:

Qual é a capital do estado com a maior população?
Resposta(c, Capital (s, c) ∧ maior (p, Estado (s) ∧ População (s, p))).

A tarefa de CHILL é aprender um predicado *Análise*(*palavras, semântica*) que seja consistente com os exemplos e, esperançosamente, generalize bem em outros exemplos. Aplicar LPI diretamente para aprender esse predicado resulta em desempenho ruim: o analisador induzido tem apenas cerca de 20% de precisão. Felizmente, os aprendizes de LPI podem melhorar adicionando conhecimento. Nesse caso, a maior parte do predicado *Análise* foi definido como um programa lógico, e a tarefa de CHILL foi reduzida a induzir as regras de controle que orientam o analisador para selecionar uma análise sobre a outra. Com esse conhecimento adicional em segundo plano, CHILL pode saber como alcançar 70-85% de precisão em várias tarefas de banco de dados.

23.3.5 Complicações

A gramática do inglês real é infinitamente complexa. Citaremos brevemente alguns exemplos.

Tempo e tempo verbal: agora, vamos supor que queiramos representar a diferença "John loves Mary" ("João ama Maria") e "John loved Mary" ("João amou Maria"). A língua inglesa utiliza tempos verbais (passado, presente e futuro) para indicar a hora relativa de um evento. Uma boa escolha para representar a hora de eventos é a notação da Seção 12.3. No cálculo de eventos, temos

John loves mary: $E_1 \in Loves(John, Mary) \wedge Durante(Agora, extensão(E_1))$
John loves mary: $E_2 \in Loves(John, Mary) \wedge Depois(Agora, extensão(E_2))$

Isso sugere que nossas duas regras léxicas para as palavras "loves" e "loved" devem ser estas:

$Verbo(\lambda x\ \lambda y\ e \in Loves(John, Mary) \wedge Durante(Agora, e)) \rightarrow$ **loves**
$Verbo(\lambda x\ \lambda y\ e \in Loves(x, y) \wedge Depois(Agora, e)) \rightarrow$ **loved.**

Exceto por essa mudança, tudo o mais sobre a gramática permanece igual, o que é uma novidade encorajadora; ela sugere que estamos no caminho certo se podemos adicionar com tanta facilidade uma complicação como os tempos verbais (embora tenhamos apenas arranhado a superfície de uma gramática completa no que se refere ao tempo e aos tempos verbais). É também encorajador que a distinção entre os processos e eventos discretos que fizemos em nossa discussão da representação do conhecimento na Seção 12.3.1 foi refletida realmente no uso da linguagem. Podemos dizer "John slept a lot last night" ("João dormiu muito a noite passada"), onde *sleeping* é uma categoria de processo, mas é estranho dizer "John found a unicorn a lot last night" ("João achou muito um unicórnio a noite passada"), onde *finding* é uma categoria de eventos discretos. Uma gramática refletiria esse fato por ter probabilidade baixa para adicionar o sintagma adverbial "a lot" para

eventos discretos.

Quantificação: considere a sentença “Every agent feels a breeze” (“Todo agente sente uma brisa”). A sentença tem apenas uma análise sintática sob ε_0, mas é realmente semanticamente ambígua; o significado preferencial é “Para todo agente existe uma brisa que ele sente”, mas um significado alternativo é “Existe um brisa que todos os agentes sentem”.[5] As duas interpretações podem ser representadas como a seguir:

$$\forall a\ a \in Agentes \Rightarrow$$
$$\quad \exists b\ \ b \in Wumpsuses \wedge \exists\ e\ \ e \in Cheira(a, b) \wedge Durante(Agora, e)\ ;$$
$$\exists b\ \ b \in Wumpsuses\ \forall a\ a \in Agentes \Rightarrow$$
$$\quad \exists\ e\ \ e \in Cheira(a, b) \wedge Durante(Agora, e).$$

A abordagem-padrão para a quantificação é que a gramática defina não uma sentença semântica lógica real, mas uma **forma quase lógica** que será, então, transformada em sentença lógica por algoritmos fora do processo de análise. Esses algoritmos podem ter regras de preferência para preferir um escopo quantificador sobre outro — preferências que não precisam ser refletidas diretamente na gramática.

Pragmática: mostramos como um agente pode perceber uma cadeia de palavras e utilizar uma gramática para derivar um conjunto de interpretações semânticas possíveis. Agora, atacaremos o problema de completar a interpretação pela adição de informações dependentes do contexto sobre a situação atual para cada interpretação candidata. A necessidade mais óbvia de informações pragmáticas está na resolução do significado de **indexicais**, que são sintagmas que se referem diretamente à situação atual. Por exemplo, na sentença “I am in Boston today”, “I” e “today” são indexicais. A palavra “I” seria representada pelo *falante* fluente e competiria ao ouvinte resolver o significado da fluência — isso não é considerado parte da gramática, mas uma questão de pragmática, de usar o contexto da situação atual para interpretar fluentes.

Outra parte da pragmática é interpretar a intenção do falante. A ação do falante é considerada um **ato de fala**, e cabe ao ouvinte decifrar qual é o tipo de ação — pergunta, afirmativa, promessa, aviso, comando, e assim por diante. Um comando como “Vá para 2 2” refere-se implicitamente ao ouvinte. Até agora, a nossa gramática para *S* cobre apenas sentenças declarativas. Podemos facilmente estendê-la para cobrir comandos. Um comando pode ser formado a partir de um *SV*, onde o sujeito é implicitamente o ouvinte. Precisamos distinguir comandos de declarações e, assim, alteramos as regras para *S*, a fim de incluir o tipo de ato da fala:

$$S(Declaração(Falante, rel(obj))) \rightarrow SN(obj)\ SV(rel)$$
$$S(Comando(Falante, rel(Ouvinte))) \rightarrow SV(rel).$$

Dependências de longa distância: as perguntas introduzem uma nova complexidade gramatical. Em “Who did the agent tell you to give the gold to?” (“Para quem o agente pediu que você desse o ouro?”), a palavra final “to” deve ser analisada como [*SP* para ␣], onde “␣” denota uma lacuna ou um **traço**, onde o *SN* está ausente; o *SN* em falta é autorizado pela primeira palavra da sentença, “who”. É utilizado um sistema de aumentos complexo para se certificar de que os *SN*s ausentes correspondem às palavras autorizadas da forma correta e proíbem lacunas nos lugares errados. Por exemplo, você não pode ter lacuna em uma ramificação de uma associação de *SN*: “What did he play

[*NP* Dungeons and ␣]?”(“O que ele jogou [*NP* Dungeons and ␣]?”) não é gramatical. Mas pode haver a mesma lacuna em ambas as ramificações de uma associação de *SV*: “What did you [*SV* [*SV* smell ␣] e [*SV* shoot na arrow at ␣]]?” (“O que você [*SV* [*SV* cheirou ␣] e [*SV* atire uma flecha em ␣]]?”)

Ambiguidade: em alguns casos, os ouvintes têm plena consciência da ambiguidade em uma expressão vocal. Aqui estão alguns exemplos em idioma inglês tirados de manchetes de jornais:

Squad helps dog bite victim.
Police began campaign to run down jaywalkers.
Helicopter powered by human flies.
Once-sagging cloth diaper industry saved by full dumps.
Portable toilet bombed; police have nothing to go on.
Teacher strikes idle kids.
Include your children when baking cookies.
Hospitals are sued by 7 foot doctors.
Milk drinkers are turning to powder.
Safety experts say school bus passengers should be belted.

Entretanto, na maior parte do tempo, a linguagem que ouvimos parece não ambígua. Desse modo, quando os primeiros pesquisadores começaram a utilizar computadores para analisar a linguagem na década de 1960, eles ficaram bastante surpresos ao perceber que *quase toda expressão vocal é altamente ambígua*, *embora as interpretações alternativas possam não ser aparentes para um falante nativo*. Um sistema com gramática e léxico extensos poderia descobrir milhares de interpretações para uma sentença perfeitamente comum. A **ambiguidade léxica**, em que uma palavra tem mais de um significado, é bastante comum; “back” pode ser um advérbio (go back), um adjetivo (back door), um substantivo (the back of the room) ou um verbo (back up your files). “Jack” pode ser um nome próprio, um substantivo (uma carta de baralho, uma alavanca, uma bandeira náutica, um peixe,um burro, uma tomada ou um guindaste para levantar objetos pesados) ou um verbo (to jack up a car, to hunt with a light ou to hit a baseball hard).

A **ambiguidade sintática** refere-se a um sintagma com análises múltiplas: “I smelled a wumpus in 2,2” tem duas análises sintáticas: uma em que o sintagma preposicional “in 2,2” modifica o substantivo e outra em que ele modifica o verbo. A ambiguidade sintática leva a uma **ambiguidade semântica** porque uma análise sintática significa que o wumpus está em 2,2 e a outra significa que há um fedor em 2,2. Nesse caso, fazer a interpretação errada pode ser um engano mortal para o agente.

Finalmente, pode haver ambiguidade entre significados literais e figurados. As figuras de estilo são importantes em poesia, mas também são surpreendentemente comuns na fala diária. Uma **metonímia** é uma figura de estilo na qual um objeto é usado para representar outro. Quando ouvimos “Chrysler announced a new model” (“A Chrysler anunciou um novo modelo”), não interpretamos essa sentença como se as empresas pudessem conversar; em vez disso, compreendemos que um porta-voz representando a empresa fez o anúncio. A metonímia é comum e, com frequência, é interpretada inconscientemente pelos ouvintes humanos. Infelizmente, a gramática inglesa escrita não é tão fácil. Para manipular a semântica da metonímia de modo apropriado, precisamos introduzir um nível de ambiguidade inteiramente novo. Isso é feito fornecendo-se *dois* objetos para a interpretação

semântica de todo sintagma na sentença: um para o objeto a que o sintagma se refere de forma literal (Chrysler) e um para a referência metonímica (o porta-voz). Em seguida, temos de dizer que existe uma relação entre os dois. Em nossa gramática atual, "Chrysler announced" é interpretada como:

$$x = Chrysler \wedge e \in Anunciou(x) \wedge Depois(Agora, extensão(e)).$$

Precisamos alterar essa equação para:

$$x = Chrysler \wedge e \in Anunciou(m) \wedge Depois(Agora, extensão(e)) \wedge Metonímia\ (m, x).$$

Isso nos diz que existe uma entidade x que é igual a Chrysler e outra entidade m que fez o anúncio, e que as duas mantêm uma relação de metonímia. A próxima etapa é definir que espécies de relações de metonímia podem ocorrer. O caso mais simples é quando não existe absolutamente nenhuma metonímia — o objeto literal x e o objeto metonímico m são idênticos:

$$\forall m, x\ (m = x)\ \Rightarrow Metonímia(m, x).$$

No caso do exemplo da Chrysler, uma generalização razoável indicaria que uma organização pode ser usada para representar um porta-voz dessa organização:

$$\forall m, x\ \ x \in Organizações \wedge Porta\text{-}voz(m, x)\ \Rightarrow Metonímia(m, x).$$

Outras metonímias incluem o autor pelas obras (eu li *Shakespeare*) ou, de modo mais geral, o produtor pelo produto (eu dirijo uma *Honda*) e a parte pelo todo (O Red Sox precisa de um *braço* forte). Alguns exemplos de metonímia, como "O *sanduíche de presunto* da mesa 4 quer outra cerveja", são mais inovadores e interpretados em relação a uma situação.

Uma **metáfora** é outra figura de estilo em que um sintagma com um único significado literal é usado para sugerir um significado diferente por meio de uma analogia. Assim, a metáfora pode ser vista como uma espécie de metonímia, onde a relação é de similaridade.

A **desambiguação** é o processo de recuperar o significado mais provável pretendido de um enunciado. Em certo sentido, já temos um quadro para resolver esse problema: cada regra tem uma probabilidade associada a ela, então a probabilidade de uma interpretação é o produto das probabilidades das regras que levaram à interpretação. Infelizmente, as probabilidades refletem como os sintagmas são comuns no *corpus* de onde a gramática foi aprendida e, portanto, refletem o conhecimento geral, não o conhecimento específico da situação atual. Para realizar desambiguação corretamente, precisamos combinar quatro modelos:

1. O **modelo do mundo**: a probabilidade de que uma proposição ocorra no mundo. Dado o que sabemos acerca do mundo, é mais provável que um falante que diz "I'm dead" ("Eu estou morto") queira dizer "I am in big trouble" ("Eu estou em apuros"), em vez de "My life ended, and yet I can still talk" ("Minha vida acabou e eu ainda posso falar").
2. O **modelo mental**: a probabilidade de que o falante forme a intenção de comunicar certo fato ao ouvinte. Essa abordagem combina modelos do que o falante acredita, o que o falante acredita que o ouvinte acredita, e assim por diante. Por exemplo, quando um político diz "I'm not a crook" ("Eu não sou trapaceiro/um cajado"), o modelo de mundo pode atribuir uma

probabilidade de apenas 50% para a proposição de que o político não seja trapaceiro e 99,999% para a proposição de que ele não seja um cajado. No entanto, podemos selecionar a interpretação anterior porque tem mais a dizer.

3. O **modelo de linguagem**: a probabilidade de que certa cadeia de palavras seja escolhida, dado que o falante tem a intenção de comunicar determinado fato.
4. O **modelo acústico:** para a comunicação falada, a probabilidade de que será gerada uma sequência especial de sons, uma vez que o falante tenha escolhido determinada cadeia de palavras.

23.4 TRADUÇÃO AUTOMÁTICA

A tradução automática é a tradução de texto de uma língua natural (a fonte) para outra (o destino). Foi uma das primeiras áreas de aplicação previstas para computadores (Weaver, 1949), mas somente na década passada a tecnologia teve seu uso espalhado de forma ampla.

Aqui está uma passagem do início deste livro em sua versão original em inglês:

AI is one of the newest fields in science and engineering. Work started in earnest soon after World War II, and the name itself was coined in 1956. Along with molecular biology, AI is regularly cited as the "field I would most like to be in" by scientists in other disciplines.
(A IA é um dos campos mais recentes da ciência e da engenharia. O trabalho começou para valer logo após a Segunda Guerra Mundial, e o próprio nome foi cunhado em 1956. Junto com a biologia molecular, a IA é regularmente citada como o "campo onde eu mais gostaria de estar" por cientistas de outras disciplinas.)

E aqui está a tradução de inglês para dinamarquês por uma ferramenta on-line, o Google Translate:

AI er en af de nyeste områder inden for videnskab og teknik. Arbejde startede for alvor lige efter Anden Verdenskrig, og navnet i sig selv var opfundet i 1956. Sammen med molekylær biologi, er AI jævnligt nævnt som "feltet Jeg ville de fleste gerne være i" af forskere i andre discipliner.

Para quem não entende dinamarquês, aqui está o dinamarquês traduzido de volta para o inglês. As palavras que apareceram diferente estão em itálico:

AI is one of the newest fields *of* science and engineering. Work *began* in earnest *just* after the *Second* World War, and the name itself was *invented* in 1956. *Together* with molecular biology, AI is *frequently mentioned* as ␣ "field I would most like to be in" by *researchers* in other disciplines.

As diferenças são todas paráfrases razoáveis, tal como *frequently mentioned* em vez de *regularly cited*. O único erro real é a omissão do artigo *the*, indicado pelo símbolo ␣. Isso é precisão típica: das duas sentenças, uma tem um erro que não teria sido feito por um falante nativo, mas o significado foi transmitido claramente.

Historicamente, houve três principais aplicações da tradução automática. A *tradução tosca*, como a prevista por serviços on-line gratuitos, fornece a "essência" de uma sentença ou documento estrangeiro, mas contém erros. A *tradução pré-editada* é utilizada por empresas para publicar sua documentação e materiais de vendas em vários idiomas. O texto-fonte original está escrito em uma linguagem limitada que é mais fácil de traduzir automaticamente, e os resultados são editados geralmente por um ser humano para corrigir erros eventuais. A *tradução com restrição de origem* funciona de forma totalmente automática, mas apenas em linguagem altamente estereotipada, como um boletim meteorológico.

Tradução é difícil porque, no caso mais geral, requer conhecimeto profundo do texto. Isso é verdade mesmo para os textos muito simples — "textos" de uma palavra. Considere a palavra "*Open*" ("Aberta") em uma grande faixa com dizeres na parte de fora de uma loja construída recentemente.[6] Significa que a loja está agora em operação diária, mas os leitores desse sinal não se sentiriam enganados se a loja fechasse durante a noite sem que a faixa fosse retirada. Os dois sinais utilizam a palavra idêntica para transmitir significados diferentes. Em alemão, o sinal na porta seria "*Offen*", enquanto na faixa seria lido "*Neu Eröffnet*".

O problema é que linguagens diferentes categorizam o mundo de forma diferente. Por exemplo, a palavra francesa "*doux*" abrange ampla gama de significados, correspondendo aproximadamente às palavras em inglês "*soft*" ("suave"), "*sweet*" ("doce") e "*gentle*" ("gentil"). Da mesma forma, a palavra em inglês "*hard*" abrange praticamente todos os usos da palavra alemã "*hart*" (fisicamente recalcitrante, cruel) e alguns usos da palavra "*schwierig*" (difícil). Portanto, representar o significado de uma sentença é mais difícil para a tradução do que é para a compreensão única do idioma. Um sistema de análise em inglês poderia usar predicados como $Open(x)$, mas para a tradução a linguagem da representação teria de fazer mais distinções, talvez com $Open_1(x)$ representando o sentido de "*Offen*" e $Open_2(x)$ representando o sentido de "*Neu Eröffnet*". **Interlíngua** é a representação da linguagem que faz todas as distinções necessárias para um conjunto de línguas.

Um tradutor (humano ou máquina), muitas vezes precisa compreender a situação real descrita na origem, e não apenas as palavras individuais. Por exemplo, para a tradução da palavra inglesa "*him*" ("lhe", "a ele") em coreano a escolha deve ser feita entre a forma humilde e a honorífica, uma escolha que depende da relação social entre o falante e o referente de "*him*". Em japonês, os honoríficos são relativos, então a escolha depende das relações sociais entre o falante, o referente e o ouvinte. Os tradutores (tanto máquina como humanos), por vezes, acham difícil fazer essa escolha. Em outro exemplo, para traduzir "*The baseball hit the window. It broke*" para o francês, devemos escolher o feminino "*elle*"ou o masculino "*il*" para "*it*"(ele ou ela em inglês), por isso temos de decidir se "*it*" refere-se ao beisebol ou à janela. Para obter a tradução correta, é preciso entender a física, bem como a linguagem.

Às vezes *não há escolha* que possa render uma tradução completamente satisfatória. Por exemplo, um poema de amor em italiano que utiliza o masculino "*il sole*" (sol) e o feminino "*la luna*" (lua) para simbolizar dois amantes terá de ser necessariamente alterado quando traduzido para o alemão, no qual os gêneros são invertidos, e ainda mais alterado quando traduzido para uma linguagem em que os gêneros são os mesmos.[7]

23.4.1 Sistemas de tradução automática

Todos os sistemas de tradução devem modelar os idiomas de origem e de destino, mas os sistemas variam no tipo de modelos que utilizam. Alguns sistemas tentam analisar todo o texto do idioma de origem em uma representação do conhecimento interlíngua e geram sentenças na linguagem-alvo a partir dessa representação. Isso é difícil porque envolve três problemas não resolvidos: criação de uma representação do conhecimento completa de tudo; análise dessa representação; e geração de sentenças dessa representação.

Outros sistemas são baseados em **modelo de transferência**. Eles mantêm um banco de dados de regras de tradução (ou exemplos) e, sempre que a regra (ou exemplo) combinam, traduzem diretamente. A transferência pode ocorrer no nível lexical, sintático ou semântico. Por exemplo, uma regra estritamente sintática faz o mapeamento do inglês [adjetivo substantivo] para o francês [substantivo adjetivo]. A mistura da regra sintática e lexical faz o mapeamento do francês [S_1 "*et puis*" S_2] para o inglês [S_1 "*and then*" S_2]. A Figura 23.12 faz um diagrama dos vários pontos de transferência.

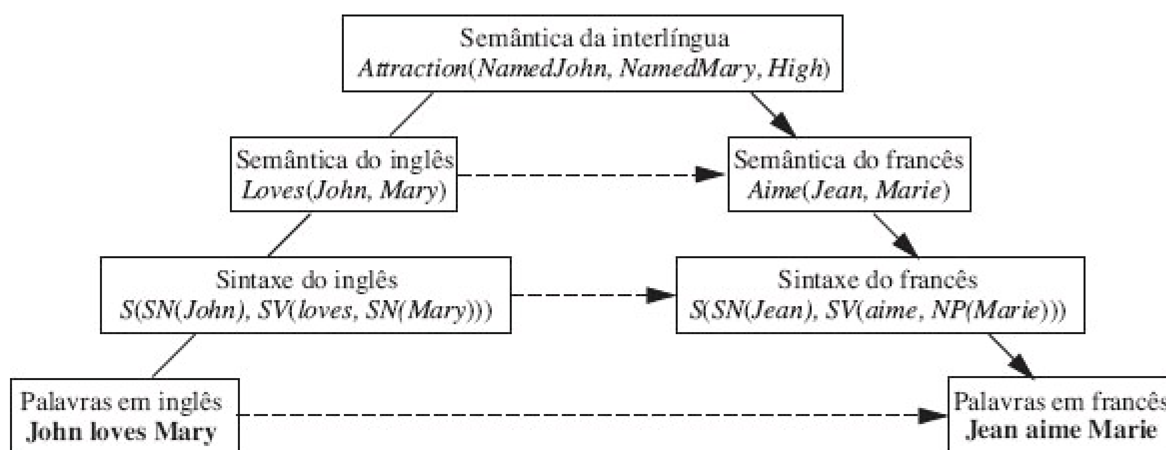


Figura 23.12 O triângulo de Vauquois: diagrama semântico esquemático das escolhas de um sistema de tradução automático (Vauquois, 1968). Inicia-se com um texto em inglês no topo. Um sistema baseado em interlíngua segue as linhas contínuas, analisando primeiro o inglês em uma forma sintática, em seguida em uma representação semântica e em uma representação interlíngua, e depois através da geração de uma forma semântica, sintática e lexical em francês. Um sistema baseado em transferência utiliza as linhas tracejadas como atalho. Sistemas diferentes fazem a transferência em pontos diferentes: alguns o fazem em vários pontos.

23.4.2 Tradução automática estatística

Agora que vimos como pode ser complexa a tarefa de tradução, não deve ser surpresa que os sistemas de tradução automática de maior sucesso são construídos pelo treinamento de um modelo probabilístico utilizando estatísticas recolhidas a partir de um *corpus* de texto grande. Essa abordagem não precisa de uma ontologia complexa de conceitos interlíngua nem precisa de gramáticas artesanais das linguagens de origem e destino, nem de uma floresta sintática rotulada à mão. Tudo o que precisa são de dados — amostras de traduções de onde um modelo de tradução

pode ser aprendido. Para traduzir uma sentença, digamos, do inglês (*e*) para o francês (*f*), encontramos a cadeia de palavras *f** que maximiza

$$f^* = \underset{f}{\operatorname{argmax}}\, P(f \mid e) = \operatorname{argmax}\, P(e \mid f)\, P(f).$$

Aqui o fator $P(f)$ é o **modelo do idioma** alvo para o francês; informa a probabilidade de dada sentença em francês. $P(e \mid f)$ é o **modelo de tradução**, que informa a probabilidade de que uma sentença em inglês seja a tradução de determinada sentença em francês. Da mesma forma, $P(f \mid e)$ é um modelo de tradução do inglês para o francês.

Devemos trabalhar diretamente em $P(f \mid e)$ ou aplicar a regra de Bayes e trabalhar em $P(e \mid f)\, P(f)$? Em aplicações de **diagnóstico** como a medicina, é mais fácil modelar o domínio na direção causal: *P*(*sintomas* | *doença*), em vez de *P*(*doença* | *sintomas*). Mas, na tradução, as duas direções são igualmente fáceis. Os primeiros trabalhos de tradução automática estatística aplicaram a regra de Bayes — em parte porque os pesquisadores tinham um modelo bom de linguagem, $P(f)$, e queriam fazer uso dele, em parte porque eles vinham de um conhecimento em reconhecimento de voz, que *é* um problema de diagnóstico. Seguimos a sua orientação neste capítulo, mas notamos que os trabalhos recentes em tradução automática estatística, muitas vezes otimizam o $P(f \mid e)$ diretamente, utilizando um modelo mais sofisticado que leva em conta muitas das características do modelo de linguagem.

O modelo de linguagem, $P(f)$, poderia abordar qualquer nível no lado direito da Figura 23.12, mas a abordagem mais fácil e mais comum é a de construir um modelo de *n*-grama a partir de um *corpus* em francês, como vimos antes. Isso apreende apenas uma ideia local parcial de sentenças em francês; no entanto, muitas vezes é suficiente para traduções toscas.[8]

O modelo de tradução é aprendido a partir de um ***corpus* bilíngue** — uma coleção de textos paralelos, cada par inglês/francês. Agora, se tivéssemos um *corpus* infinitamente grande, traduzir uma sentença seria apenas uma tarefa de pesquisa: teríamos visto a sentença em inglês antes no *corpus*, assim poderíamos apenas retornar a sentença francesa paralela. Mas é claro que nossos recursos são finitos, e a maioria das sentenças a serem solicitadas para traduzir será recente. No entanto, serão compostas de sintagmas que já vimos antes (mesmo que alguns sintagmas sejam tão curtos como uma palavra). Por exemplo, neste livro, sintagmas comuns incluem “neste exercício vamos”, “tamanho do espaço de estados”, “em função do” e “notas no final do capítulo”. Se solicitado para traduzir a sentença nova “Neste exercício, vamos calcular o tamanho do espaço de estados como uma função do número de ações” para o francês, devemos ser capazes de quebrar a sentença em sintagmas, encontrar os sintagmas no *corpus* em inglês (neste livro), encontrar os sintagmas franceses correspondentes (a partir da tradução francesa do livro) e depois remontar os sintagmas franceses em uma ordem que faça sentido em francês. Em outras palavras, dada uma sentença cuja fonte é inglês *e*, encontrar um tradução francesa *f* é uma questão de três etapas:

1. Quebrar a sentença em inglês em sintagmas $e_1, \ldots, e_n$.
2. Para cada sintagma e_i, escolher um sintagma em francês correspondente f_i. Usamos a notação $P(f_i \mid e_i)$ para a probabilidade frasal de que f_i seja uma tradução de e_i.
3. Escolher uma permutação da frases $f_1 \ldots, f_n$. Vamos especificar essa permutação de uma forma

que parece um pouco complicada, mas é projetada para ter uma distribuição simples de probabilidade: para cada f_i, escolhemos uma **distorção** d_i, que é o número de palavras que o sintagma f_i moveu com relação a f_{i-1}; positivo se moveu para a direita, negativo se moveu para a esquerda e zero se f_i seguiu imediatamente f_{i-1}.

A Figura 23.13 mostra um exemplo do processo. No topo, a sentença "Há um wumpus fedorento dormindo em 2 2" é dividida em cinco sintagmas, $e_1, \ldots, e_5$. Cada um deles será traduzido em um sintagma f_1 correspondente, e então eles serão permutados pela ordem f_1, f_3, f_4, f_2, f_5. Especificaremos a permutação em termos de distorções d_i de cada sintagma em francês, definido como

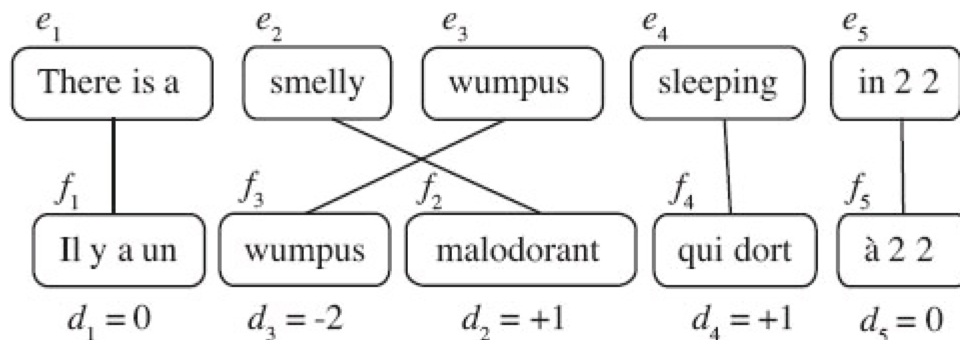


Figura 23.13 Sintagmas franceses candidatos para cada sintagma de uma sentença em inglês, com valores de distorção (d) para cada sintagma em francês.

$$d_i = \text{INÍCIO}\,(f_i) - \text{FIM}\,(f_{i-1}) - 1,$$

onde INÍCIO(f_i) é o número ordinal da primeira palavra do sintagma f_i na sentença em francês e FIM(f_{i-1}) é o número ordinal da última palavra do sintagma f_{i-1}. Na Figura 23.13 observamos que f_5, "*à 2 2*" segue imediatamente f_4, "*qui dort*"" e, assim, $d_5 = 0$. Entretanto, o sintagma f_2 moveu uma palavra à direita de f_1, assim $d_2 = 1$. Como caso especial temos $d_1 = 0$, porque f_1 inicia na posição 1 e FIM(f_0) é definido como 0 (embora f_0 não exista).

Agora que definimos a distorção, d_i, podemos definir a distribuição de probabilidade para a distorção, $\mathbf{P}(d_i)$. Observe que, para sentenças limitadas pelo comprimento n temos $|d_i| \leq n$ e de modo que o total de distribuição de probabilidade $\mathbf{P}(d_i)$ tem apenas $2n + 1$ elementos, muito menos números para aprender do que os números de permutações, $n!$. É por isso que definimos a permutação nesse caminho tortuoso. Claro, esse é um modelo bastante empobrecido de distorção. Não informa que os adjetivos são geralmente distorcidos para aparecer após o substantivo quando estamos traduzindo do inglês para o francês — esse fato é representado no modelo francês de língua, $P(f)$. A probabilidade da distorção é completamente independente das palavras nos sintagmas — depende apenas do valor inteiro d_i. A distribuição de probabilidade fornece um resumo da volatilidade das permutações; qual a probabilidade de uma distorção de $P(d = 2)$, em comparação com $P(d = 0)$, por exemplo.

Estamos prontos agora para juntar tudo: podemos definir $P(f, d \mid e)$, a probabilidade de que a sequência de sintagmas f com distorções d é uma tradução da sequência de sintagmas e. Fizemos a

suposição de que cada tradução de sintagma e cada distorção é independente das outras, e, assim, podemos fatorar a expressão como

$$P(f, d \mid e) = \prod_i P(f_i \mid e_i)\, P(d_i) \cdot$$

Isso oferece uma maneira de calcular a probabilidade $P(f, d \mid e)$ para uma tradução candidata f e uma distorção d. Mas, para encontrar os melhores f e d não podemos apenas enumerar sentenças; talvez com 100 sintagmas em francês para cada sintagma em inglês no *corpus* existam 100^5 traduções diferentes de 5-sintagmas e 5! reordenações para cada uma delas. Teremos de procurar uma boa solução. Uma busca de feixe local com heurística que estima a probabilidade provou ser eficaz em encontrar a tradução mais provável.

Tudo o que resta é aprender as probabilidades frasais e de distorção. Esboçamos o procedimento; consulte as notas no final do capítulo para obter detalhes.

1. **Encontrar textos paralelos**: em primeiro lugar, reúna um *corpus* paralelo bilíngue. Por exemplo, um **Hansard**[9] é um registro do debate parlamentar. Canadá, Hong Kong e outros países produzem hansards bilíngues, a União Europeia publica seus documentos oficiais em 11 idiomas, e as Nações Unidas publicam documentos multilíngues. O texto bilíngue também está disponível on-line, alguns sites publicam conteúdo com URLs em paralelo, por exemplo, `/en/` para a página em inglês e `/fr/` para a página correspondente francesa. Os sistemas de tradução estatística principais experimentam em centenas de milhões de palavras de textos em paralelo e bilhões de palavras do texto monolíngue.
2. **Segmento em sentenças**: a unidade de tradução é uma sentença, então teremos de quebrar o *corpus* em sentenças. Períodos são fortes indicadores do final de uma sentença, mas considere "Dr. J. R. Smith de Rodeo Dr. paid $29.99 on 9.9.09."; somente o ponto final termina a sentença. Uma maneira de decidir se um ponto termina uma sentença é analisar um modelo que tome como características as palavras ao redor e suas partes da fala. Essa abordagem atinge cerca de 98% de precisão.
3. **Alinhar sentenças**: determinar a quais sentenças corresponde na versão francesa cada sentença na versão em inglês. Geralmente, a próxima sentença em inglês corresponde à próxima sentença em francês em uma combinação de 1:1, mas às vezes há uma variação: uma sentença em um idioma será dividida em uma combinação de 2:1 ou a ordem de duas sentenças será trocada, resultando em uma combinação de 2:2. Ao olhar para o comprimento da sentença em si (isto é, sentenças curtas deveriam se alinhar com sentenças curtas), é possível alinhá-las (1:1, 1:2 ou 2:2 etc.) com precisão na faixa de 90-99% usando uma variação do algoritmo de Viterbi. Pode também ser conseguido um alinhamento melhor através de marcos que são comuns em ambos os idiomas, tais como números, datas, nomes próprios ou palavras que, de um dicionário bilíngue, sabemos que têm tradução inequívoca. Por exemplo, se a terceira sentença em inglês e a quarta em francês contiverem a sequência "1989" e as sentenças vizinhas não tiverem, é uma boa evidência de que as sentenças deveriam ser alinhadas em conjunto.
4. **Alinhar sintagmas**: dentro de uma sentença, os sintagmas podem ser alinhados por um processo que é semelhante ao que é utilizado para o alinhamento de sentença, mas requer melhoria

iterativa. Quando começamos, não temos nenhuma maneira de saber que "*qui dort*" alinha-se com "*sleeping*", mas podemos chegar a esse alinhamento por um processo de agregação de evidências. Em todas as sentenças de exemplo que vimos, percebemos que "*qui dort*" e "*sleeping*" coocorrem com alta frequência e que no par de sentenças alinhadas nenhum sintagma diferente de "*qui dort*" coocorre com tanta frequência em outras frases com "*sleeping*". Um alinhamento completo do sintagma sobre o nosso *corpus* nos dá as probabilidades frasais (após o alisamento adequado).

5. **Extrato de distorções**: uma vez que temos um alinhamento de sintagmas podemos definir as probabilidades de distorção. Basta contar quantas vezes a distorção ocorre no *corpus* para cada distância $d = 0, \pm 1, \pm 2, \ldots$ e aplicar o alisamento.
6. **Melhorar as estimativas com EM**: utilizar expectativa de maximização para melhorar as estimativas dos valores $P(f \mid e)$ e $P(d)$. Calculamos os melhores alinhamentos com os valores atuais desses parâmetros na etapa E, em seguida atualizamos as estimativas na etapa M e iteramos o processo até a convergência.

23.5 RECONHECIMENTO DE VOZ

O **reconhecimento de voz** é a tarefa de identificar uma sequência de palavras proferidas por um falante, dado um sinal acústico. Tornou-se uma das aplicações principais de IA — milhões de pessoas interagem com os sistemas de reconhecimento de voz a cada dia para navegar por sistemas de correio de voz, pesquisar na Web a partir de telefones móveis e outras aplicações. A voz é uma opção atraente quando há necessidade de operação com as mãos livres e ao operar máquinas.

O reconhecimento de voz é difícil porque os sons feitos por um falante são ambíguos e… ruidosos. Como exemplo bem conhecido, o sintagma "*recognize speech*" soa quase o mesmo que "*wreck a nice beach*", quando falado rapidamente. Esse exemplo curto mostra vários dos problemas que tornam a voz problemática. Primeiro, a **segmentação**: as palavras escritas em inglês têm espaços entre elas, mas na fala rápida não há pausas em "*wreck a nice*" que a distinguisse como uma frase de várias palavras em oposição à palavra "*recognize*". Segundo, a **coarticulação**: quando se fala rapidamente, o som do "s" ao final de "*nice*" se funde com o som de "b" do início de "*beach*", produzindo algo como "sp". Outro problema que não aparece nesse exemplo é o de **homófonos** — palavras como "*to*", "*too*" e "*two*", que têm o mesmo som, mas diferem em significado.

Podemos ver o reconhecimento de voz como um problema na explicação da sequência mais provável. Como vimos na Seção 15.2, esse é o problema de calcular a sequência mais provável das variáveis de estado, $\mathbf{x}_{1:t}$, dada uma sequência de observações $\mathbf{e}_{1:t}$. Nesse caso, as variáveis de estado são as palavras, e as observações são os sons. Mais precisamente, uma observação é um vetor de características extraído do sinal de áudio. Como de costume, a sequência mais provável pode ser calculada com a ajuda da regra de Bayes:

$$\underset{palavra_{1:t}}{argmax}\, P(palavra_{1:t} \mid som_{1:t}) = \underset{palavra_{1:t}}{argmax}\, P(som_{1:t} \mid palavra_{1:t})\, P(palavra_{1:t}).$$

Aqui $P(som_{1:t} \mid palavra_{1:t})$ é o **modelo acústico**. Descreve os sons das palavras — que "**ceiling**"

(teto) começa com um “c” suave e soa como “*sealing*” (vedação). P($palavra_{1:t}$) é conhecido como **modelo de linguagem**. Especifica a probabilidade antes de cada emissão — por exemplo, que “*ceiling fan*” (ventilador de teto) tem cerca de 500 vezes mais probabilidade como sequência de palavras que “*sealing fan*” (ventilador de vedação).

Essa abordagem foi nomeada por Claude Shannon (1948) de **modelo de canal ruidoso**. Ele descreveu uma situação em que uma mensagem original (as *palavras* em nosso exemplo) é transmitida através de um canal ruidoso (como uma linha de telefone) de tal forma que uma mensagem alterada (os *sons* no nosso exemplo) é recebida na outra ponta. Shannon mostrou que não importa o quão ruidoso seja o canal, é possível recuperar a mensagem original com erro arbitrariamente pequeno se codificarmos a mensagem original de forma bastante redundante. A abordagem de canal ruidoso tem sido aplicada para reconhecimento de voz, tradução automática, correção ortográfica e outras tarefas.

Uma vez que definimos os modelos acústicos e de linguagem, podemos resolver, para a sequência de palavras mais provável, utilizando o algoritmo de Viterbi (Seção 15.2.3). A maioria dos sistemas de reconhecimento de voz usa um modelo de linguagem que realiza a suposição de Markov — que o estado atual $Palavra_t$ depende apenas de um número fixo n de estados anteriores — e representa $Palavra_t$ como uma única variável aleatória que assume um conjunto finito de valores, que a torna um Modelo Oculto de Markov (MOM). Assim, o reconhecimento de voz torna-se uma aplicação simples da metodologia do MOM, como descrito na Seção 15.3 — simples, uma vez definidos os modelos acústico e de linguagem. Falaremos sobre ele em seguida.

23.5.1 Modelo acústico

As ondas sonoras são mudanças periódicas de pressão que se propagam pelo ar. Quando essas ondas incidem no diafragma de um microfone, o movimento de vaivém gera uma corrente elétrica. Um conversor analógico-digital mede o tamanho da corrente — que se aproxima da amplitude das ondas sonoras — em intervalos discretos chamado de **taxa de amostragem**. Sons de voz, que estão na sua maioria na faixa de 100 Hz (100 ciclos por segundo) a 1.000 Hz, são amostrados tipicamente a uma taxa de 8 kHz (CDs e arquivos de mp3 são amostrados a 44,1 kHz).

A precisão de cada medida é determinada pelo **fator de quantização**; os reconhecedores de voz normalmente mantêm 8-2 bits. Isso significa que um sistema de baixo custo, com amostragem de 8 kHz, com quantização de 8 bits, iria requerer quase metade de um megabyte por minuto de fala.

Uma vez que só queremos saber que palavras foram ditas, não exatamente como soaram, não precisamos manter toda essa informação. Precisamos apenas fazer a distinção entre sons de vozes diferentes. Os linguistas identificaram cerca de 100 sons de voz, ou **fones**, ou segmentos de fala mínimos, que podem ser compostos para formar todas as palavras em todas as línguas humanas conhecidas. Grosseiramente falando, um fone é o som que corresponde a uma única vogal ou consoante, mas existem algumas complicações: combinações de letras, como “th” e “ng” produzem um fone único, e algumas letras produzem fones diferentes em contextos diferentes (por exemplo, o “a” em *rat* e *rate*. A Figura 23.14 lista todos os fonemas que são usados em inglês, com um exemplo

de cada. Um **fonema** é a menor unidade de som que tem significado distinto para os falantes de uma língua em particular. Por exemplo, o "t" em "*stick*" soa bem similar ao "t" em "*tick*" que os falantes de inglês consideram o mesmo fonema. Mas a diferença é significativa no idioma tailandês, por isso há dois fonemas. Para representar o inglês falado precisamos de uma representação que pode distinguir entre fonemas diferentes, mas que não precisa distinguir as variações não fonéticas do som: voz alta ou baixa, rápida ou lenta, masculina ou feminina etc.

Vogais		Consoantes B–N		Consoantes P–Z	
Telefone	Exemplo	Telefone	Exemplo	Telefone	Exemplo
[iy]	beat	[b]	bet	[p]	pet
[ih]	bit	[ch]	Chet	[r]	rat
[eh]	bet	[d]	debt	[s]	set
[æ]	bat	[f]	fat	[sh]	shoe
[ah]	but	[g]	get	[t]	ten
[ao]	bought	[hh]	hat	[th]	thick
[ow]	boat	[hv]	high	[dh]	that
[uh]	book	[jh]	jet	[dx]	butter
[ey]	bait	[k]	kick	[v]	vet
[er]	Bert	[l]	let	[w]	wet
[ay]	buy	[el]	bottle	[wh]	which
[oy]	boy	[m]	met	[y]	yet
[axr]	diner	[em]	bottom	[z]	zoo
[aw]	down	[n]	net	[zh]	measure
[ax]	about	[en]	button		
[ix]	roses	[ng]	sing		
[aa]	cot	[eng]	washing	[-]	*silence*

Figura 23.14 O alfabeto fonético ARPA, ou **ARPAbet**, listando todos os fones usados em inglês americano. Existem várias notações alternativas, incluindo a Internacional Phonetical Alphabet (IPA), que contém os fones em todas as línguas conhecidas.

Em primeiro lugar, observamos que, embora as frequências de som de voz possam ter vários kHz, as *mudanças* no conteúdo do sinal ocorrem com muito menos frequência, talvez em não mais de 100 Hz. Portanto, os sistemas de voz resumem as propriedades do sinal em intervalos de tempo chamados **quadros**. O comprimento de um quadro de cerca de 10 milissegundos (ou seja, 80 amostras em 8 kHz) é curto o suficiente para garantir que poucos fenômenos de curta duração serão perdidos. Os quadros sobrepostos são utilizados para nos certificarmos de não perder um sinal por acontecer de cair sobre um limite do quadro.

Cada quadro é resumido por um vetor de **características**. Escolher as características de um sinal de voz é como ouvir uma orquestra e dizer: "Aqui as trompas estão tocando bem alto e os violinos suavemente." Apresentaremos uma breve descrição das características de um sistema típico. Primeiro, utiliza-se uma transformada de Fourier para determinar a quantidade de energia acústica em cerca de uma dúzia de frequências. Em seguida, calculamos uma medida chamada de **coeficiente de frequência mel-cepstral** (**CFMC**) ou **CFMC** para cada frequência. Calculamos também o total de energia no quadro. Isso fornece 13 características; para cada uma calculamos a diferença entre esse

quadro e o anterior, e a diferença entre as diferenças, para um total de 39 características. Esses são de valor contínuo; a maneira mais fácil de adequá-los à estrutura MOM é discretizar os valores (também é possível estender o MOM para manusear misturas contínuas de gaussianas). A Figura 23.15 mostra a sequência de transformações do som em estado natural para uma sequência de quadros com características distintas.

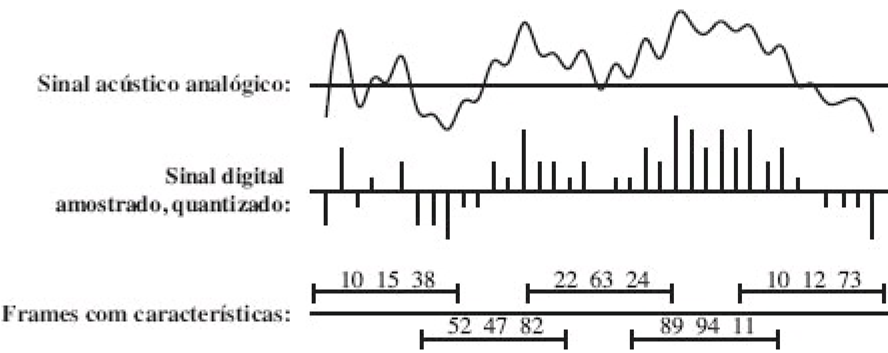


Figura 23.15 Tradução do sinal acústico em uma sequência de quadros. Nesse diagrama, cada quadro é descrito pelos valores discretizados de três características acústicas; um sistema real teria dezenas de características.

Vimos como ir do sinal acústico em estado natural a uma série de observações, $\mathbf{e}_t$. Agora, temos de descrever os estados (não observáveis) do MOM e definir a transição do modelo, $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1})$, e o modelo de sensor, $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$. O modelo de transição pode ser quebrado em dois níveis: palavra e fone. Vamos começar pela parte inferior: o **modelo do fone** descreve um fone como três estados, o início, o meio e o fim. Por exemplo, o fone [t] tem início em silêncio, um pequeno rompante explosivo de som ao meio, e (geralmente) um assobio no final. A Figura 23.16 mostra um exemplo para o fone [m]. Observe que, na fala normal, um fone médio tem duração de 5-10 milissegundos ou 5-10 frames. Em cada estado, os autolaços permitem variações nesse período. Com muitos autolaços (especialmente no estado do meio), podemos representar um longo som "mmmmmmmmmmm". Ignorar os autolaços produz um som "m" curto.

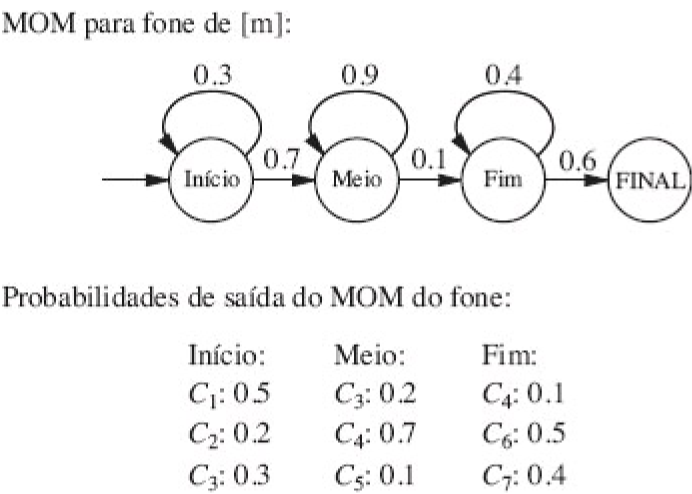
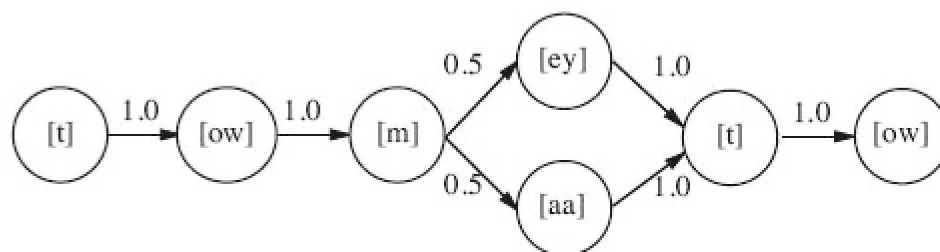


Figura 23.16 Um MOM para o fone de três estados [m]. Cada estado tem várias saídas possíveis, cada uma com sua própria probabilidade. Os rótulos das características de C_1 a C_7 do CFMC são arbitrários, sustentando algumas combinações de valores característicos.

Na Figura 23.17, os modelos de fone são amarrados para formar um **modelo de pronúncia** para uma palavra. De acordo com Gershwin (1937), você diz [t ow m ey t ow] e eu digo [t ow m aa t ow]. A Figura 23.17(a) mostra um modelo de transição que prevê essa variação de dialeto. Cada um dos círculos nesse diagrama representa um modelo de fone como o da Figura 23.16.



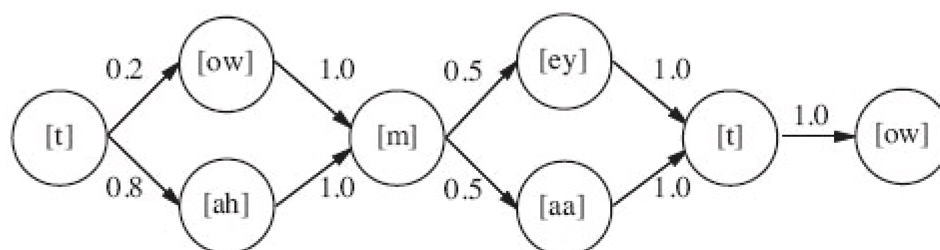


Figura 23.17 Dois modelos de pronúncia da palavra "tomato". Cada modelo é mostrado como um diagrama de transição com estados como círculos e setas indicando as transições permitidas com as suas probabilidades associadas. (a) Modelo que permite diferenças de dialeto. Os números 0,5 são estimativas baseadas nas pronúncias preferidas de cada um dos dois autores. (b) Modelo com efeito de coarticulação sobre a primeira vogal, permitindo o fone [ow] ou [ah].

Além da variação de dialeto, as palavras podem ter variação de **coarticulação**. Por exemplo, o fone [t] é produzido com a língua na parte superior da boca, enquanto o [ow] tem a língua perto da parte de baixo. Ao falar rapidamente, a língua não tem tempo para ficar em posição para o [ow], e acabamos com [t ah] em vez de [t ow]. A Figura 23.17(b) apresenta um modelo para "tomato", que leva esse efeito de coarticulação em conta. Modelos de fones mais sofisticados levam em conta o contexto dos fones circundantes.

Pode haver variação substancial na pronúncia de uma palavra. A pronúncia mais comum de "because" é [b iy k ah z], mas isso representa apenas cerca de um quarto das utilizações. O outro quarto (aproximadamente) substitui [ix], [ih] ou [ax] pela primeira vogal, e o restante substitui [ax] ou [aa] pela segunda vogal, [zh] ou [s] pelo final [z] ou tira o "b" inteiramente, deixando "cuz".

23.5.2 Modelo de linguagem

Para propósitos gerais de reconhecimento de voz, o modelo de linguagem pode ser um modelo *n*-grama de texto instruído a partir de um *corpus* de sentenças escritas. No entanto, a linguagem falada tem características diferentes da linguagem escrita, por isso é melhor obter um *corpus* de transcrições de linguagem falada. Para a tarefa específica de reconhecimento de voz, o *corpus* deve

ser uma tarefa específica: a construção de um sistema de reservas aéreas, obtenção de transcrições de chamadas anteriores. Ele também ajuda a ter vocabulário de tarefas específicas, como uma lista de todos os aeroportos e cidades atendidas, e todos os números de voo.

Parte do projeto de uma interface de usuário de voz é para coagir o usuário a dizer coisas de um conjunto limitado de opções, para que o reconhecedor de voz lide com uma distribuição de probabilidade mais compacta. Por exemplo, a pergunta "A qual cidade você quer ir?" provoca uma resposta com um modelo de linguagem altamente restrito, enquanto a pergunta "Como posso ajudá-lo?", não.

23.5.3 Construindo um reconhecedor de voz

A qualidade de um sistema de reconhecimento de voz depende da qualidade de todos os seus componentes — o modelo de linguagem, os modelos de pronúncia de palavras, os modelos de fone e os algoritmos processadores de sinais utilizados para extrair características espectrais do sinal acústico. Discutimos como o modelo de linguagem pode ser construído a partir de um *corpus* de texto escrito e deixamos os detalhes de processamento de sinal para outros livros. Ficamos com a pronúncia e os modelos de fones. A *estrutura* dos modelos de pronúncia — tal como o modelo tomate na Figura 23.17 — geralmente é desenvolvido manualmente. Já estão disponíveis grandes dicionários de pronúncia de inglês e de outros idiomas, apesar de a sua precisão variar muito. A estrutura dos modelos de fone de três estados é o mesmo para todos os fones, como mostrado na Figura 23.16. Isso deixa as probabilidades por si só.

Como de costume, vamos adquirir as probabilidades de um *corpus*, dessa vez um *corpus* de fala. O tipo mais comum de *corpus* para se obter é aquele que inclui o sinal de voz para cada sentença emparelhada com uma transcrição de palavras. Construir um modelo a partir desse *corpus* é mais difícil do que construir um modelo de texto de *n*-grama porque temos de construir um modelo oculto de Markov — a sequência de fones para cada palavra e o estado do fone para cada frame de tempo são variáveis ocultas. Nos primórdios do reconhecimento de fala, as variáveis ocultas eram fornecidas pela laboriosa rotulagem à mão de espectrogramas. Os sistemas recentes utilizam a maximização de expectativa para fornecer automaticamente os dados em falta. A ideia é simples: dado um MOM e uma sequência de observação, podemos utilizar os algoritmos de alisamento das Seções 15.2 e 15.3 para calcular a probabilidade de cada estado em cada passo de tempo e, por uma simples extensão, a probabilidade de cada estado — pares de estados em intervalos de tempo consecutivos. Essas probabilidades podem ser vistas como *rótulos incertos*. Dos rótulos incertos, podemos estimar a nova transição e as probabilidades de sensor, e o procedimento de EM se repete. É garantido que o método aumente o ajuste entre o modelo e os dados em cada iteração, e geralmente convirja para um conjunto muito melhor de valores de parâmetros do que os previstos pelas estimativas iniciais, rotuladas à mão.

Os sistemas com a precisão mais alta trabalham com o treinamento de um modelo diferente para cada falante, assim capturando as diferenças de dialeto, bem como masculino/feminino e outras variações. Esse treinamento pode requerer várias horas de interação com o falante; assim, os sistemas com a adoção mais difundida não criam modelos de falantes específicos.

A precisão de um sistema depende de uma série de fatores. Primeiro, importa a qualidade do sinal: um microfone direcional de alta qualidade destinado a uma boca estacionária em um quarto acolchoado vai funcionar bem melhor do que um microfone barato transmitindo um sinal através de linhas telefônicas de um carro em tráfego com o rádio tocando. O tamanho do vocabulário importa: ao reconhecer cadeias de dígitos com um vocabulário de 11 palavras (1-9 mais "oh" e "zero"), a taxa de erro por palavra estará abaixo de 0,5%, enquanto sobe cerca de 10% com novas histórias e um vocabulário de 20.000 palavras, e 20% com um *corpus* com um vocabulário de 64.000 palavras. A tarefa também importa: quando o sistema está tentando realizar uma tarefa específica — reservar um voo ou dar instruções em um restaurante —, a tarefa muitas vezes pode ser realizada perfeitamente, mesmo com taxa de erro de palavra de 10% ou mais.

23.6 RESUMO

A compreensão da linguagem natural é um dos subcampos mais importantes da IA. Ela se baseia em ideias da filosofia e da linguística, bem como em técnicas de representação do conhecimento lógico e probabilístico e de raciocínio. Diferentemente de outras áreas da IA, a compreensão da linguagem natural exige uma investigação empírica do comportamento humano real — algo complexo e interessante.

- A teoria da linguagem formal e as gramáticas de **estrutura de sintagma** (e, em particular, a gramática **livre de contexto**) são ferramentas úteis para lidar com alguns aspectos da linguagem natural. O formalismo da gramática livre de contexto probabilística (GLCP) é amplamente utilizado.
- As sentenças em uma linguagem livre de contexto podem ser analisadas sintaticamente no tempo $O(n^3)$ por um **analisador de diagramas**, como o **algoritmo CYK**, que requer regras gramaticais para estar na **forma normal de Chomsky**.
- Uma **floresta sintática** pode ser utilizada para aprender uma gramática. Também é possível aprender uma gramática de um *corpus* de sentenças não analisadas, mas isso não é tão bem-sucedido.
- Uma **GLCP lexicalizada** nos permite representar que algumas relações entre as palavras são mais comum do que outras.
- É conveniente **aumentar** uma gramática para tratar de problemas como a concordância entre sujeito e verbo e o caso pronominal. A **gramática de cláusulas definidas** (GCD) é um formalismo que permite aumentos. Com a GCD, a análise e a interpretação semântica (e até mesmo a geração) podem ser realizadas com o uso de inferência lógica.
- A **interpretação semântica** também pode ser manipulada por uma gramática aumentada. Uma forma quase lógica pode ser um intermediário útil entre as árvores sintática e semântica.
- A **ambiguidade** é um problema muito importante na compreensão da linguagem natural; a maioria das sentenças tem muitas interpretações possíveis, mas, em geral, apenas uma é apropriada. A eliminação da ambiguidade se baseia no conhecimento sobre o mundo, sobre a situação atual e sobre o uso da linguagem.
- Os sistemas de **tradução automática** foram implementados usando uma variedade de técnicas,

da análise sintática e semântica completa até técnicas estatísticas com base em frequências de sintagmas. Atualmente, os modelos estatísticos são mais populares e mais bem-sucedidos.

- Os sistemas de **reconhecimento de voz** também foram baseados principalmente em princípios estatísticos. Sistemas de voz são populares e úteis, ainda que imperfeitos.
- Juntos, tradução automática e reconhecimento de fala são dois dos grandes sucessos da tecnologia da linguagem natural. Uma das razões pela qual os modelos têm bom desempenho é que estão disponíveis grandes *corpora* — tanto a tradução como a fala são tarefas que são executadas "no meio natural" por pessoas no dia a dia. Em contraste, tarefas como análise de sentenças têm tido menos sucesso, em parte porque não está disponível grande *corpora* de sentenças analisadas "ao natural" e em parte porque a análise em si mesma não é útil.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Como redes semânticas, gramáticas livres de contexto (também conhecidas como gramáticas de estrutura sintagmática) são uma reinvenção de uma técnica usada pela primeira vez pelos antigos gramáticos indianos (sobretudo Panini, c. 350 a.C.), estudando sânscrito shátrico (Ingerman, 1967). Elas foram reinventadas por Noam Chomsky (1956) para a análise da sintaxe inglesa e independentemente por John Backus para a análise da sintaxe Algol-58. Peter Naur estendeu a notação de Backus e agora é creditado (Backus, 1996) com o "N" na BNF, que originalmente significava "Backus Normal Form". Knuth (1968) definiu um tipo de gramática aumentada chamada **gramática de atributo** que é útil para linguagens de programação. As gramáticas de cláusulas definidas foram introduzidas por Colmerauer (1975) e desenvolvidas e popularizadas por Pereira e Shieber (1987).

As **gramáticas livres de contexto probabilísticas** foram investigadas por Booth (1969) e Salomaa (1969). Outros algoritmos para GLCPs foram apresentados na excelente e curta monografia de Charniak (1993) e os excelentes e longos livros de Manning e Schütze (1999) e Jurafsky e Martin (2008). Baker (1979) apresenta o algoritmo dentro-fora para aprender um GLCP, e Lari e Young (1990) descrevem seus usos e limitações. Stolcke e Omohundro (1994) mostram como aprender regras gramaticais fundindo com o modelo bayesiano; Haghighi e Klein (2006) descrevem um sistema de aprendizagem com base em protótipos.

GLCPs lexicalizadas (Charniak, 1997; Hwa, 1998) combinam os melhores aspectos das GLCPs e modelos de *n*-grama. Collins (1999) descreve a análise de GLCP que é lexicalizada com as características principais. Petrov e Klein (2007a) mostram como obter as vantagens da lexicalização sem os aumentos lexicais reais pela instrução de categorias específicas de aprendizagem da floresta sintática que tem categorias gerais; por exemplo, a floresta sintática tem uma categoria de *SN*, de onde pode ser aprendidas categorias mais específicas, como SN_O e SN_S.

Houve inúmeras tentativas de escrever gramáticas formais das línguas naturais, tanto em linguística "pura" como computacional. Existem várias gramáticas abrangentes, mas informais de inglês (Quirk *et al.*, 1985; McCawley, 1988; Huddleston e Pullum, 2002). A partir de meados da década de 1980, desenvolveu-se uma tendência no sentido de incluir mais informações no léxico e menos na gramática. A gramática funcional léxica, ou LFG (*lexical-functional grammar*) (Bresnan,

1982), foi o primeiro formalismo gramatical importante a se concentrar intensamente no léxico. Se levarmos a lexicalização a um extremo, acabaremos chegando à **gramática de categorias** (Clark e Curran, 2004), na qual pode haver até mesmo (apenas) duas regras gramaticais, ou à **gramática de dependência** (Smith e Eisner, 2008; Kübler *et al*., 2009), em que não existe nenhuma categoria sintática, somente relações entre palavras. Sleator e Temperley (1993) descrevem um analisador sintático de dependência. Paskin (2001) mostra que uma versão de gramática de dependência é mais fácil de aprender que GLCPs.

Os primeiros algoritmos computadorizados de análise sintática foram demonstrados por Yngve (1955). Algoritmos eficientes foram desenvolvidos no final da década de 1960, com algumas mudanças de rumo desde então (Kasami, 1965; Younger, 1967; Graham *et al*., 1980). Maxwell e Kaplan (1993) mostram como a análise por diagrama com aumentos pode se tornar eficiente no caso médio. Church e Patil (1982) estudam a resolução da ambiguidade sintática. Klein e Manning (2003) descrevem a análise sintática de A*, e Pauls e Klein (2009) estendem-na para análise *K*-best A*, em que o resultado não é uma única análise, mas o melhor *K*.

As análises sintáticas líderes de hoje incluem as de Petrov e Klein (2007b), que alcançaram 90,6% de precisão sobre o *corpus* do *Wall Street Journal*, Charniak e Johnson (2005) alcançaram 92,0%, e Koo *et al*. (2008) alcançaram 93,2% sobre o Treebank Penn. Esses números não são diretamente comparáveis, e há algumas críticas sobre o campo que está se concentrando muito estreitamente em poucos *corpora* seletos e talvez superadaptando sobre eles.

A interpretação semântica formal de linguagens naturais teve origem na filosofia e na lógica formal, particularmente no trabalho de Alfred Tarski (1935) sobre a semântica de linguagens formais. Bar-Hillel (1954) foi o primeiro a considerar os problemas da pragmática e a propor a possibilidade de tratá-los por meio da lógica formal. Por exemplo, ele introduziu o termo *indexical* de C. S. Peirce (1902) na linguística. O ensaio de Richard Montague "English as a formal language" (1970) é uma espécie de manifesto da análise lógica da linguagem, mas o livro de Dowty *et al*. (1991) e Portner e Partee (2002) são de melhor leitura.

O primeiro sistema de PLN a resolver uma tarefa real foi provavelmente o sistema de resposta a perguntas BASEBALL (Green *et al*., 1961), que lidava com perguntas sobre um banco de dados de estatísticas de beisebol. Pouco depois foi lançado o LUNAR de Woods (1973), que respondia a perguntas sobre as pedras lunares trazidas da Lua pelo programa Apollo. Roger Schank e seus alunos construíram uma série de programas (Schank e Abelson, 1977; Schank e Riesbeck, 1981) na qual todos os programas tinham a tarefa de compreender a linguagem. Abordagens modernas para interpretação semântica geralmente assumem que o mapeamento da sintática para a semântica será aprendido de exemplos (Zelle e Mooney, 1996; Zettlemoyer e Collins, 2005).

Hobbs *et al*. (1993) descrevem uma estrutura quantitativa não probabilística de interpretação. Trabalhos mais recentes seguem uma estrutura explicitamente probabilística (Charniak e Goldman, 1992; Wu, 1993; Franz, 1996). Em linguística, a teoria do caráter ótimo (Kager, 1999) se baseava na ideia de elaboração de restrições suaves na gramática, dando origem a uma graduação natural das interpretações (similar a uma distribuição de probabilidade), em vez de fazer a gramática gerar todas as possibilidades com grau idêntico. Norvig (1988) descreve os problemas relacionados à consideração de várias interpretações simultâneas, em vez de se fixar em uma única interpretação de probabilidade máxima. Os críticos literários (Empson, 1953; Hobbs, 1990) têm sido ambíguos

quanto ao fato de a ambiguidade ser algo a ser resolvido ou apreciado.

Nunberg (1979) traça um modelo formal da metonímia. Lakoff e Johnson (1980) apresentam uma análise atrativa e catalogam metáforas comuns em inglês. Martin (1990) e Gibbs (2006) ofereceram modelos computacionais de interpretação de metáforas.

O primeiro resultado importante em **indução de gramática** foi negativo: Gold (1967) mostrou que não é possível aprender de modo confiável uma gramática livre de contexto correta, dado um conjunto de cadeias dessa gramática. Linguistas proeminentes, como Chomsky (1957) e Pinker (2003), usaram o resultado de Gold para argumentar que tem de existir uma **gramática universal** inata que todas as crianças possuem desde o nascimento. O argumento conhecido como *Poverty of the Stimulus* afirma que não é oferecido às crianças insumo suficiente para aprender a GLC, portanto elas já "conhecem" a gramática e têm de afinar meramente alguns de seus parâmetros. Embora esse argumento continue a influenciar grande parte da linguística de Chomsky, ele foi rejeitado por alguns outros linguistas (Pullum, 1996; Elman *et al*., 1997) e pela maioria dos cientistas da computação. Já em 1969, Horning mostrou que *é* possível aprender, no sentido do aprendizado PAC, uma gramática probabilística livre de contexto. Desde então, houve muitas demonstrações empíricas convincentes de aprendizado a partir somente de exemplos positivos, como o trabalho de PLI de Mooney (1999) e o de Muggleton e De Raedt (1994), a aprendizagem em sequência de Nevill-Manning e Witten (1997), e as notáveis teses de doutorado de Schütze (1995) e Marcken (1996). Há a International Conference on Grammatical Inference (ICGI) anualmente. É possível aprender outros formalismos gramaticais, como linguagens regulares (Denis, 2001) e autômatos de estados finitos (Parekh e Honavar, 2001). Abney (2007) é um livro introdutório à aprendizagem semissupervisionada de modelos de linguagem.

O Wordnet (Fellbaum, 2001) é um dicionário à disposição do público com cerca de 100.000 palavras e sintagmas, categorizado em partes de fala e ligado por relações semânticas, como sinônimo, antônimo e merônimo. O Treebank Penn (Marcus *et al*., 1993) fornece árvores de análise de um *corpus* em inglês de três milhões de palavras. Charniak (1996) e Klein e Manning (2001) discutem análise com gramáticas de floresta sintática. O British National Corpus (Leech *et al*., 2001) contém 100 milhões de palavras, e a World Wide Web contém vários trilhões de palavras; Brants *et al*. (2007) descrevem modelos de *n*-grama de um *corpus* com dois trilhões de palavras na Web.

Na década de 1930, Petr Troyanskii fez um pedido de patente para uma "máquina de tradução", mas não havia computadores disponíveis para implementar suas ideias. Em março de 1947, a Fundação Rockefeller de Warren Weaver escreveu a Norbert Wiener, sugerindo que uma máquina para tradução poderia ser possível. Valendo-se do trabalho em criptografia e teoria da informação, Weaver escreveu: "Quando vejo um artigo em russo, eu digo: 'Isso realmente está escrito em inglês, mas foi codificado com símbolos estranhos. Vou agora passar a decodificar'." Na década seguinte, a comunidade tentou decodificar dessa forma. A IBM exibiu um sistema rudimentar em 1954. Bar-Hillel (1960) descreve o entusiasmo desse período. No entanto, o governo dos Estados Unidos informou posteriormente (ALPAC, 1966) que "não há perspectiva imediata ou previsível de tradução automática útil". No entanto, o trabalho limitado continuou e, começando na década de 1980, o poder do computador aumentou até o ponto em que os resultados da ALPAC não eram mais corretos.

A abordagem estatística básica que descrevemos no capítulo é baseada no trabalho precoce pelo grupo da IBM (Brown *et al*., 1988, 1993) e nos trabalhos recentes do ISI e de pesquisa do grupo

Google (Och e Ney, 2004;. Zollmann *et al*., 2008). Koehn (2009) forneceu um livro introdutório sobre tradução automática estatística, e também foi muito influente um pequeno tutorial por Kevin Knight (1999). Os primeiros trabalhos sobre segmentação de sentença foram feitos por Palmer e Hearst (1994). Och e Ney (2003) e Moore (2005) cobrem o alinhamento de sentença bilíngue.

A pré-história de reconhecimento de voz começou nos anos 1920 com o Rádio Rex, um cachorro de brinquedo que se movia por ativação de voz. Rex pulava para fora da sua casinha em resposta à palavra "Rex" (ou, na verdade, qualquer palavra suficientemente forte). Um trabalho um pouco mais sério começou após a Segunda Guerra Mundial. No Bell Labs da AT&AT, foi construído um sistema para o reconhecimento de dígitos isolados (Davis *et al*., 1952), por meio de correspondência de padrão simples de características acústicas. A partir de 1971, a Defense Advanced Research Projects Agency (DARPA) do Departamento de Defesa dos Estados Unidos financiou quatro projetos concorrentes de cinco anos para desenvolver sistemas de alto desempenho de reconhecimento de voz. O vencedor, e o único sistema a atingir a meta de 90% de precisão com um vocabulário de 1.000 palavras, foi o sistema HARPY em CMU (Lowerre e Reddy, 1980). A versão final do HARPY foi derivada de um sistema chamado DRAGON construído pelo estudante James Baker (1975) da CMU; o DRAGON foi o primeiro a usar MOMs para a fala. Quase simultaneamente, Jelinek (1976) na IBM desenvolveu outro sistema baseado em MOM. Os últimos anos foram caracterizados pelo progresso incremental constante, maiores conjuntos de dados e modelos, e competições mais rigorosas sobre tarefas de voz mais realistas. Em 1997, Bill Gates previu: "Daqui a cinco anos, você não reconhecerá o PC porque a voz virá na interface." Isso não se concretizou, mas em 2008 ele previu: "Em cinco anos, a Microsoft tem a expectativa de que sejam realizadas mais buscas na Internet por meio da fala do que através de digitação em um teclado." A história dirá se ele está certo dessa vez.

Muitos bons livros sobre o reconhecimento de voz estão disponíveis (Rabiner e Juang, 1993; Jelinek, 1997; Ouro e Morgan, 2000;. Huang *et al*., 2001). A apresentação neste capítulo valeu-se da pesquisa realizada por Kay, Gawron e Norvig (1994) e do livro de Jurafsky e Martin (2008). Uma pesquisa sobre reconhecimento de voz foi publicada em *Computer Speech and Language, Speech Communications*, no IEEE *Transactions on Acoustics, Speech, and Signal Processing*,e nos workshops de DARPA sobre o processamento de voz e a linguagem natural e nas conferências do Eurospeech, ICSLP e ASRU.

Ken Church (2004) mostra que a pesquisa de linguagem natural tem um ciclo entre a concentração em dados (empirismo) e a concentração em teorias (racionalismo). O linguista John Firth (1957) proclamou: "Você conhecerá uma palavra pela associação que ela mantém"; e a linguística dos anos 1940 e início dos anos 1950 foi baseada principalmente em frequência de palavras, embora sem o poder computacional que temos disponível hoje. Em seguida, Noam (Chomsky, 1956) mostrou as limitações dos modelos de estado finito e provocou interesse pelos estudos teóricos da sintaxe, desconsiderando a contagem de frequência. Essa abordagem dominou por 20 anos, até que o empirismo fez um retorno baseado no sucesso do trabalho no reconhecimento estatístico da fala (Jelinek, 1976). Hoje, a maioria dos trabalhos aceita a estrutura estatística, mas há grande interesse na construção de modelos estatísticos que consideram modelos de alto nível, tais como relações entre árvores sintáticas e semânticas, não apenas sequências de palavras.

Foram apresentados trabalhos sobre aplicações de processamento de linguagem na conferência

bienal Applied Natural Language Processing (ANLP), na conferência sobre métodos empíricos Empirical Methods in Natural Language Processing (EMNLP) e no periódico *Natural Language Engineering*. Uma gama ampla de trabalho de PNL aparece na revista *Computational Linguistics* e sua conferência, ACL, e na conferência de *Computational Linguistics* (COLING).

EXERCÍCIOS

23.1 Leia o texto a seguir uma vez para compreendê-lo e memorizar o máximo que puder. Haverá um teste mais adiante.

> O procedimento é realmente bastante simples. Primeiro, você organiza os itens em diferentes grupos. É claro que uma pilha pode ser suficiente, dependendo do quanto existe para fazer. Se tiver de ir para algum outro lugar devido à falta de recursos, essa será a próxima etapa; do contrário, você estará muito bem adaptado. É importante não se exceder. Ou seja, é melhor fazer poucas coisas de cada vez do que muitas. Em curto prazo, isso pode não parecer importante, mas podem surgir complicações facilmente. Um equívoco também é dispendioso. A princípio, o procedimento inteiro parecerá complicado. Porém, logo ele se tornará apenas outro fato da vida. É difícil prever qualquer propósito para a necessidade dessa tarefa no futuro imediato, mas nunca se sabe quando vai ser preciso realizá-la. Depois que o procedimento for concluído, o material será organizado de novo em diferentes grupos. Em seguida, eles poderão ser colocados em seus lugares apropriados. Eventualmente, os itens serão usados mais uma vez e o ciclo inteiro terá de ser repetido. Contudo, isso faz parte da vida.

23.2 Uma *gramática de MOM* é essencialmente um padrão de MOM, cuja variável de estado é N (não terminal, com valores como *Det*, *Adjetivo*, *Substantivo*, e assim por diante) e cuja variável de evidência é W (palavra, com valores como *é, pato*, e assim por diante). O MOM inclui um modelo anterior $\mathbf{P}(N_0)$, um modelo de transição $\mathbf{P}(N_{t+1} \mid N_t)$ e um modelo de sensor $\mathbf{P}(W_t \mid N_t)$. Mostre que toda a gramática do MOM pode ser escrita como GLPC. [Sugestão: comece a pensar em como o MOM anterior pode ser representado por regras GLPC para o símbolo da sentença. Pode ser que seja útil ilustrar o MOM particular com os valores A e B para N e os valores x, y para W.]

23.3 Considere o seguinte GLCP para **sintagmas verbais** simples:

0.1 : *SV* → *V erbo*

0.2 : SV → *Adjetivo cópula.*

0.5 : *SV* → *Substantivo verbalizado*

0.2 : SV → *SV Advérbio*

0.5 : *Verbo* → *está*

0.5 : *Verbo* → *atira*

0.8 :*Cópula* → *é*

0.2 : *Cópula* → *parece*

0.5 : *Adjetivo* → **não bem**

0.5 : *Adjetivo* → **bem**

0.5 : *Advérbio* → **bem**

0.5 : *Advérbio* → **mal**

0.6 : *Substantivo* → **pato**

0.4 : *Substantivo* → **bem**

a. Qual dos seguintes tem probabilidade não zero como o SV? (i) atirar no pato bem bem bem (ii) parece o bem bem (iii) atira no não bem bem mal

b. Qual a probabilidade de gerar "está bem bem"?

c. Que tipos de ambiguidades são exibidas pela sentença em (b)?

d. Dado qualquer GLPC, é possível calcular a probabilidade que o GLPC gere uma sequência de 10 palavras exatamente?

23.4 Descreva as principais diferenças entre Java (ou qualquer outra linguagem de computador com a qual você está familiarizado) e o idioma inglês, fazendo comentários sobre o problema de "compreensão" em cada caso. Pense em itens como gramática, sintaxe, semântica, pragmática, composicionalidade, dependência do contexto, ambiguidade léxica, ambiguidade sintática, localização de referências (inclusive pronomes), conhecimento prático e, em primeiro lugar, o que significa "compreender".

23.5 Este exercício se refere às gramáticas que correspondem a linguagens muito simples.

a. Escreva uma gramática livre de contexto para a linguagem $a^n b^n$.

b. Escreva uma gramática livre de contexto para a linguagem de palíndromos: o conjunto de todas as cadeias cuja segunda metade é o inverso da primeira metade.

c. Escreva uma gramática sensível ao contexto para a linguagem duplicada: o conjunto de todas as cadeias cuja segunda metade é igual à primeira metade.

23.6 Considere a sentença "Someone walked slowly to the supermarket" ("Alguém caminhou lentamente para o supermercado") e o léxico consistindo nas seguintes palavras:

Pronome → **someone**

Adv → **slowly**

Det → **the**

V → **walked**

Prep → **to**

Substantivo → **supermarket**

Qual das três gramáticas a seguir, combinada com o léxico, gera a sentença dada? Mostre a(s) árvore(s) de análise correspondente(s).

(A):	(B):	(C):
$S \rightarrow SN\ SV$	$S \rightarrow SN\ SV$	$S \rightarrow SN\ SV$
$SN \rightarrow Pronome$	$SN \rightarrow Pronome$	$SN \rightarrow Pronome$
$SN \rightarrow Artigo\ Substantivo$	$SN \rightarrow Substantivo$	$SN \rightarrow Artigo\ SN$
$SV \rightarrow SV\ SP$	$SN \rightarrow Artigo\ SN$	$SV \rightarrow Verbo\ Adv$
$SV \rightarrow SV\ Adv\ Adv$	$SV \rightarrow Verbo\ Vmod$	$Adv \rightarrow Adv\ Adv$
$SV \rightarrow Verbo$	$Vmod \rightarrow Adv\ Vmod$	$Adv \rightarrow SP$
$SP \rightarrow Prep\ SN$	$Vmod \rightarrow Adv$	$SP \rightarrow Prep\ SN$
$SN \rightarrow Substantivo$	$Adv \rightarrow SP$	$SN \rightarrow Substantivo$
$SP \rightarrow Prep\ SN$		

Para cada uma das três gramáticas precedentes, anote três sentenças em inglês e três sentenças que não estejam em inglês geradas pela gramática. Cada sentença deve ser significativamente distinta, ter pelo menos seis palavras e incluir algumas novas entradas lexicais (que você deve definir). Sugira meios para melhorar cada gramática, a fim de evitar gerar s sentenças que não pertencem ao idioma inglês.

23.7 Reúna alguns exemplos de expressão de tempo, tais como "duas horas," "meia noite," e "12:46." Selecione também alguns exemplos que não sejam gramaticais, tais como "treze horas" ou "duas e quarenta e cinco." Escreva uma gramática para a linguagem do tempo.

23.8 Neste exercício, você vai transformar ε_0 na Forma Normal de Chomsky (FNC). Há cinco etapas: (a) Adicione um novo símbolo de início, (b) Elimine as regras, (c) elimine múltiplas palavras nos lados direitos, (d) Elimine as regras da forma ($X \rightarrow Y$), (e) Converta os lados direitos longos em regras binárias.

a. O símbolo de início, *S*, pode ocorrer apenas no lado esquerdo em um FNC. Adicione uma regra nova da forma $S' \rightarrow$ S, usando um novo símbolo S'.

b. A cadeia vazia não pode aparecer no lado direito em FNC. ε_0 não tem nenhuma regra com ϵ assim, isso não é problema.

c. Uma palavra pode aparecer no lado direito em uma única regra da forma ($X \rightarrow palavra$). Substitua cada regra da forma ($X \rightarrow \ldots palavra \ldots$) por ($X \rightarrow \ldots W' \ldots$) e ($W' \rightarrow palavra$) usando um novo símbolo W'.

d. A regra *($X \rightarrow Y$) não é permitida* no FNC; ela deve ser ($X \rightarrow Y Z$) ou *($X \rightarrow palavra$)*. Substitua cada regra da forma ($X \rightarrow$ Y) com um conjunto de regras da forma ($X \rightarrow \ldots$), uma para cada regra ($Y \rightarrow \ldots$), onde (…) indica um ou mais símbolos.

e. Substitua cada regra da forma ($X \rightarrow Y Z \ldots$) com duas regras, ($X \rightarrow Y Z'$) e ($Z' \rightarrow Z \ldots$), onde Z' é um novo símbolo.

Mostre cada etapa do processo e o conjunto final de regras.

23.9 Usando a notação GCD, escreva uma gramática para uma linguagem que seja semelhante a ε_1, exceto pelo fato de impor a concordância entre o sujeito e o verbo de uma sentença e, desse modo, não gerar sentenças que não sejam gramaticais como "I smells the wumpus".

23.10 Considere o seguinte GLCP:

$S \rightarrow SV\ SN$ [1.O]

$SV \rightarrow Nome$ [0.6] | *Pronome* [0.4]

$SV \rightarrow Verbo\ SN$ [0.8] | *Verbo modal* [0.2]

Nome → **posso** [0.1] **pescar** [0.3] | …

Pronome → **Eu** [0.4] | …

Verbo → **posso** [0.01] | **pescar** [0.1] | …

Modal → **posso** [0.3] | …

A sentença "Eu posso pescar" tem duas árvores de análise com essa gramática. Mostre as duas árvores, suas probabilidades prévia e condicional. dada a sentença

23.11 Uma gramática aumentada livre de contexto pode representar idiomas que uma gramática regular livre de contexto não pode. Mostre uma gramática aumentada livre de contexto para a linguagem $a^n\ b^n\ c^n$. Os valores permitidos para o aumento das variáveis são 1 e SUCESSOR(n), onde n é um valor. A regra nesse idioma para uma sentença é

$$S(n) \rightarrow A(n)\ B(n)\ C(n).$$

Mostrar a regra para *A*, *B e C.*

23.12 Aumente a gramática £1 para que ela trate de acordo artigo-substantivo. Ou seja, certifique-se que "agentes" e "um agente'' são SNs, mas "agente" e "um agentes" não são.

23.13 Considere a seguinte sentença (do *The New York Times*, 28 de julho de 2008):

> Banks struggling to recover from multibillion-dollar loans on real estate are curtailing loans to American businesses, depriving even healthy companies of money for expansion and hiring.
>
> (Bancos em dificuldades de muitos bilhões de dólares para se recuperar de empréstimos de imóveis estão reduzindo os empréstimos às empresas americanas, privando de dinheiro até mesmo as empresas saudáveis para expansão e contratação.)

a. Qual das palavras são lexicalmente ambíguas?

b. Encontre dois casos de ambiguidade sintática (há mais de dois.)

c. Dê um exemplo de metáfora.

d. Você consegue encontrar ambiguidade semântica?

23.14 Sem olhar de novo o Exercício 23.1, responda às perguntas a seguir:

a. Quais são as quatro etapas mencionadas?

b. Qual etapa foi omitida?

c. Qual é "o material" mencionado no texto?

d. Que tipo de equívoco seria dispendioso?

e. É melhor fazer muito pouco ou muito? Por quê?

23.15 Selecione cinco sentenças e as submeta a um serviço de tradução on-line. Traduza-as do inglês

para outra língua e de volta para o inglês. Classifique as sentenças resultantes com respeito à gramaticalidade e preservação de significado. Repita o processo; a segunda rodada de iteração traz resultados piores ou os mesmos resultados? Será que a escolha da linguagem intermediária faz diferença na qualidade dos resultados? Se você conhece um idioma estrangeiro, examine a tradução de um parágrafo do idioma. Conte e descreva os erros cometidos, e conclua por que esses erros foram feitos.

23.16 Os valores D_i para a sentença na Figura 23.13 somam 0. Isso vai ser verdade para todos os pares de tradução? Prove ou dê um contraexemplo.

23.17 (Adaptado de Knight, 1999.) Nosso modelo de tradução assume que, após o modelo de tradução de sintagmas selecionar sintagmas e o modelo de distorção os permutar, o modelo de linguagem poderá decodificar a permutação. Este exercício investiga como esse pressuposto é sensato. Tente decodificar essa lista de sintagmas propostos na ordem correta:

a. have, programming, a, seen, never, I, language, better

(tinha, programação, uma, visto, nunca, eu, linguagem, melhor)

b. loves, john, mary

(ama, joão, maria)

c. is the, communication, exchange of, intentional, information brought, by, about, the production, perception of, and signs, from, drawn, a, of, system, signs, conventional, shared

(é a, comunicação, troca de, intencional, informação trazida, pelo, sobre, a produção, percepção de, e sinais, de, inferida, uma, do, sistema, sinais, convencional, compartilhado)

d. created, that, we hold these, to be, all men, truths, are, equal, self-evident

(criado, que, nós defendemos estas, ser, todos os homens, verdades, são iguais, que dispensa explicação).

Quais você poderia fazer? Que tipo de conhecimento extraiu? Treine um modelo bigrama de um *corpus* de treinamento e use-o para encontrar a maior probabilidade de permutação de algumas sentenças de um *corpus* de teste. Relate sobre a exatidão desse modelo.

23.18 Calcule o caminho mais provável através do MOM na Figura 23.16 para a saída da sequência [C_1, C_2, C_3, C_4, C_4, C_6, C_7]. Dê também sua probabilidade.

23.19 Esquecemos de mencionar que o texto do Exercício 23.1 se intitula “Lavando roupas”. Releia o texto e responda às perguntas do Exercício 23.14. Conseguiu se sair melhor dessa vez? Bransford e Johnson (1973) usaram esse texto em um experimento com melhor controle e descobriram que o título ajudou significativamente. A que conclusão você chega sobre como a linguagem e a memória funcionam?

---

[1] GLCPs também são conhecidos como gramáticas livre de contexto estocástico ou GLCEs.

[2] Uma cláusula relativa segue e modifica um sintagma nominal. Ela consiste em um pronome relativo (como “who” ou “that”) seguido por um sintagma verbal. Um exemplo de cláusula relativa é *that stinks* (*que fede*) em “The wumpus *that stinks* is in 2 2”. Outra espécie de cláusula relativa não tem pronome relativo, por exemplo, *I know* em “the man *I know*”.

[3] Também poderia ser a ambiguidade $O(c!)$ na forma como os componentes conjugam, por exemplo, (X e (Y e Z)) *versus* ((X e Y) e Z),

mas isso é outra história, bem contada por Church e Patil (1982).

[4] O caso subjetivo é chamado às vezes também de nominativo, e o caso objetivo, às vezes, é chamado de caso acusativo. Muitas línguas também têm um caso dativo para palavras na posição de objeto indireto.

[5] Se essa interpretação parece diferente, considere “Every Protestant believes in a just God”.

[6] Esse exemplo é devido a Martin Kay.

[7] Warren Weaver (1949) relata que Max Zeldner aponta que o grande poeta judeu H.N. Bialik disse uma vez que a tradução “é como beijar a noiva através do véu”.

[8] Para uma tradução com mais qualidade, certamente *n*-gramas são insuficientes. O romance de Marcel Proust de 4.000 páginas, *Em busca do tempo perdido*, começa e termina com a mesma palavra (*longtemps*), portanto, alguns tradutores decidiram fazer o mesmo, basear a tradução da palavra final que apareceu cerca de dois milhões de palavras antes.

[9] Nomeado devido a William Hansard, quem primeiro publicou os debates parlamentares britânicos em 1811.

CAPÍTULO 24

Percepção

Em que conectamos o computador ao mundo bruto, em estado natural.

A **percepção** fornece aos agentes informações sobre o mundo que eles habitam interpretando a resposta dos sensores. Um sensor mede algum aspecto do ambiente de uma forma que pode ser usada como entrada por um programa do agente. O sensor poderia ser tão simples como um interruptor, que fornece um bit informando se está ligado ou desligado, ou tão complexo como o olho. Existe uma variedade de modalidades de sensores disponíveis para agentes artificiais. Aqueles que são compartilhados com os humanos incluem a visão, a audição e o toque. As modalidades que não estão disponíveis para o ser humano sozinho incluem rádio, infravermelho, GPS e os sinais sem fio. Alguns robôs fazem **sensoriamento ativo**, o que significa que enviam um sinal, como radar ou ultrassom, e sentem o reflexo desse sinal fora do ambiente. Em vez de tentar abranger tudo, este capítulo vai cobrir uma modalidade em profundidade: a visão.

Vimos, em nossa descrição de POMDPs (Seção 17.4), que um agente de decisão baseada em modelos, em ambiente parcialmente observável, tem um **modelo de sensor** — uma distribuição de probabilidade $\mathbf{P}(E \mid S)$ sobre a evidência do que seus sensores produzem, dado um estado do mundo. A regra de Bayes pode então ser utilizada para atualizar a estimativa do estado.

Para a visão, o modelo de sensor pode ser dividido em dois componentes: um **modelo de objeto** descreve os objetos que habitam o mundo visual — pessoas, edifícios, árvores, carros etc. O modelo de objeto pode incluir um modelo geométrico preciso em 3-D, reaproveitado de um sistema CAD (desenho com auxílio de computador), ou pode ser constituído por restrições vagas, como o fato de que os olhos humanos geralmente estão 5-7 cm separados. Um **modelo de renderização** descreve processos físicos, geométricos e estatísticos que produzem o estímulo do mundo. Os modelos de renderização são bastante precisos, mas são ambíguos. Por exemplo, um objeto branco com pouca luz pode parecer da mesma cor que um objeto preto sob luz intensa. Um pequeno objeto próximo pode parecer o mesmo que um objeto a grande distância. Sem provas adicionais, não podemos dizer se a imagem que preenche o frame é um Godzilla de brinquedo ou um monstro real.

A ambiguidade pode ser gerida com conhecimento prévio — sabemos que o Godzilla não é real, então a imagem deve ser um brinquedo — ou, por escolha seletiva, ignorar a ambiguidade. Por exemplo, o sistema de visão de um carro autônomo pode não ser capaz de interpretar os objetos que estão muito distantes, mas o agente pode optar por ignorar o problema porque é improvável bater contra um objeto que está a milhas de distância.

Um agente de decisão teórico não é a única arquitetura que pode fazer uso de sensores de visão. Por exemplo, as moscas da fruta (*Drosophila*) são em parte agentes reflexos: têm fibras cervicais gigantes, que formam um caminho direto de seu sistema visual para os músculos das asas que iniciam uma escapada reflexa — uma reação imediata, sem deliberação. Moscas e muitos outros animais voadores fazem uso de uma arquitetura de controle de laço fechado para aterrissar sobre um objeto. O sistema visual extrai uma estimativa da distância até o objeto, e o sistema de controle ajusta os músculos das asas em conformidade, permitindo mudanças rápidas de direção, sem necessidade de um modelo detalhado do objeto.

Em comparação com os dados de outros sensores (como o de um único bit, que informa o robô do aspirador que ele bateu em uma parede), observações visuais são extraordinariamente ricas, tanto nos detalhes que podem revelar como na enorme quantidade de dados que produzem. Uma câmera de vídeo em aplicações de robótica pode produzir um milhão de pixels de 24 bits a 60 Hz, uma taxa de 10 GB por minuto. O problema de um agente com capacidade de visão, então, é: *que aspectos do rico estímulo visual devem ser considerados para ajudar o agente a fazer boas escolhas de ações e que aspectos devem ser ignorados?* A visão — e toda percepção — serve para favorecer os objetivos do agente, não como um fim em si.

Podemos caracterizar três grandes abordagens para o problema. A abordagem da **extração de características**, como mostrado pela *Drosophila,* enfatiza cálculos simples aplicados diretamente às observações do sensor. Na abordagem do **reconhecimento**, um agente faz distinções entre os objetos que encontra com base em informações visuais e em outras. O reconhecimento poderia significar rotular cada imagem com um sim ou um não quanto conter alimentos que devemos apanhar ou a conter o rosto da avó. Finalmente, na abordagem da **reconstrução**, um agente constrói um modelo geométrico do mundo a partir de uma imagem ou um conjunto de imagens.

Os últimos 30 anos de pesquisas produziram ferramentas e métodos poderosos para tratar essas abordagens. Compreender esses métodos requer compreensão dos processos pelos quais as imagens são formadas. Portanto, agora discorreremos sobre os fenômenos físicos e estatísticos que ocorrem na produção de uma imagem.

24.1 FORMAÇÃO DE IMAGENS

A imagem distorce a aparência dos objetos. Por exemplo, uma foto tirada olhando para um longo conjunto reto de via férrea vai sugerir que os trilhos convergem e se encontram. Outro exemplo: se você segura a sua mão na frente do olho, pode bloquear a Lua, que não é menor do que a sua mão. À medida que você move a mão para trás e para a frente ou a inclina, ela parece encolher e crescer *na imagem*, mas não na realidade (Figura 24.1). Os modelos desses efeitos são essenciais para o reconhecimento e a reconstrução.

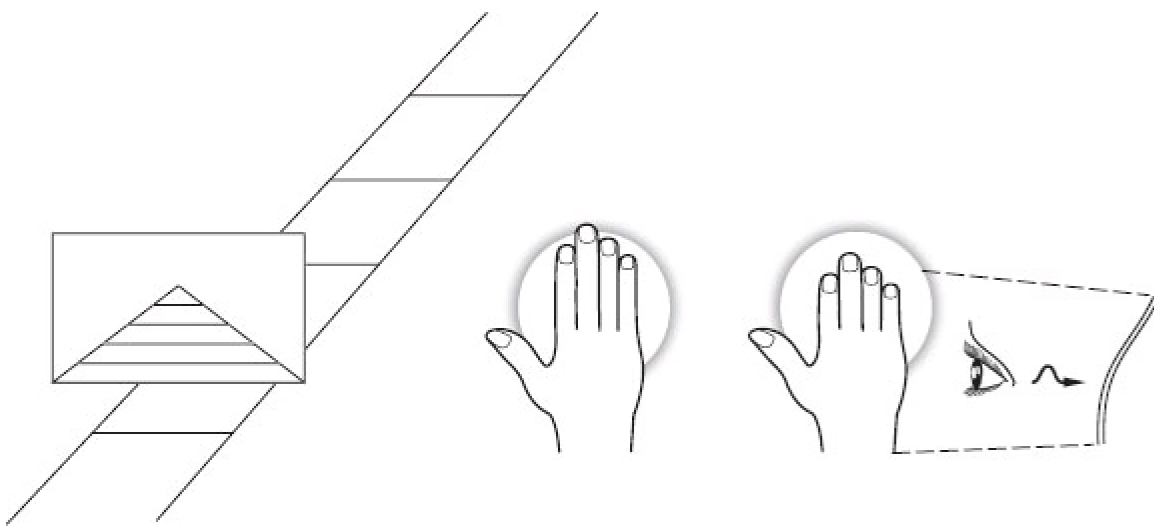


Figura 24.1 A imagem distorce a geometria. Linhas paralelas parecem reunir-se à distância, como na imagem dos trilhos da ferrovia à esquerda. No centro, uma pequena mão bloqueia a maior parte de uma Lua grande. À direita, um efeito de encurtamento: a mão é inclinada para longe do olho, fazendo com que pareça menor do que a figura central.

24.1.1 Imagens sem lentes: a câmara escura

Os sensores de imagem reúnem luz espalhada a partir de objetos em uma **cena** e criam uma **imagem** bidimensional. No olho, a imagem é formada na retina, que consiste em dois tipos de células: cerca de 100 milhões de bastonetes, que são sensíveis à luz em ampla gama de comprimentos de onda, e 5 milhões de cones. Cones, que são essenciais para a visão das cores, são de três tipos principais, cada um dos quais é sensível a um conjunto diferente de comprimentos de onda. Nas câmeras, a imagem é formada em um plano da imagem, que pode ser um pedaço de filme revestido com haletos de prata ou uma grade retangular de alguns milhões de **pixels** fotossensíveis, cada um semicondutor de óxido metálico complementar (CMOS) ou dispositivo emparelhado para carregamento (CCD). Cada fóton que chega ao sensor produz um efeito cuja força depende do comprimento de onda do fóton. A saída do sensor é a soma de todos os efeitos, devido aos fótons observados em alguma janela de tempo, o que significa que os sensores de imagem relatam uma média ponderada da intensidade da luz que chega ao sensor.

Para ver uma imagem focada, temos de assegurar que todos os fótons aproximadamente do mesmo local na cena cheguem a aproximadamente o mesmo ponto no plano de imagem. A maneira mais simples de formar uma imagem é usar uma **câmara escura**, que consiste em um orifício feito por um alfinete, *O*, na parte frontal de uma caixa, e um plano de imagem na parte traseira da caixa (Figura 24.2). Os fótons da cena devem passar pelo orifício; por isso, se for pequeno o suficiente, os fótons próximos à cena estarão nas proximidades do plano da imagem, e a imagem estará em foco.

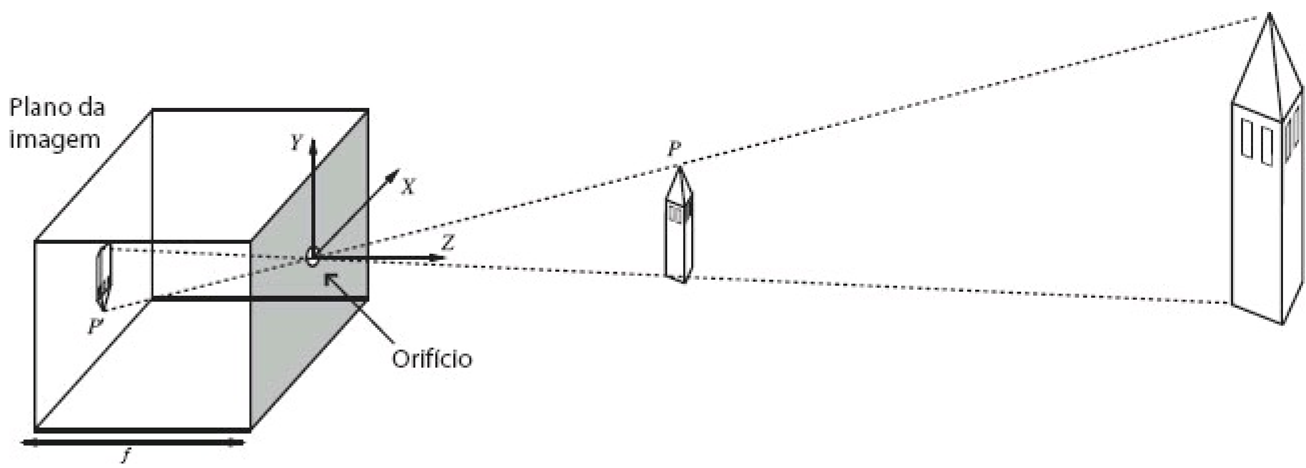


Figura 24.2 Cada elemento sensível à luz no plano de imagem na parte de trás de uma câmara escura recebe a luz de uma gama pequena de direções que passa através do orifício. Se o orifício for pequeno o suficiente, o resultado é uma imagem focada na parte de trás do orifício. O processo de projeção significa que objetos grandes e distantes parecem os mesmos que objetos menores e mais próximos. Observe que a imagem é projetada de cabeça para baixo.

A geometria da cena e da imagem é mais fácil de entender com a câmara escura. Usaremos um sistema de coordenadas tridimensional com a origem na câmera escura, e consideraremos um ponto P na cena, com as coordenadas (X, Y, Z). P é projetado no ponto P' no plano da imagem com coordenadas (x, y, z). Se f é a distância do orifício até o plano da imagem, por semelhança de triângulos, podemos derivar as seguintes equações:

$$\frac{-x}{f} = \frac{X}{Z}, \frac{-y}{f} = \frac{Y}{Z} \quad \Rightarrow \quad x = \frac{-fX}{Z}, y = \frac{-fY}{Z} .$$

Essas equações definem um processo de formação de imagem conhecido como **projeção em perspectiva**. Observe que Z no denominador significa que, quanto mais longe estiver um objeto, menor será sua imagem.

Observe também que os sinais de menos significam que a imagem está *invertida*, tanto no sentido horizontal quanto no vertical, em comparação com a cena.

De acordo com a projeção em perspectiva, os objetos distantes parecem pequenos. Isso é o que permite que você cubra a Lua com a mão (Figura 24.1). Um resultado importante desse efeito é que as linhas paralelas convergem para um ponto no horizonte (pense em vias férreas, Figura 24.1). Uma linha na cena passando na direção (U, V, W) e passando pelo ponto (X_0, Y_0, Z_0) pode ser descrita como o conjunto de pontos $(X_0 + \lambda U, Y_0 + \lambda V, Z_0 + \lambda W)$, com λ variando entre $-\infty$ e $+\infty$. Escolhas diferentes de (X_0, Y_0, Z_0) produzem linhas diferentes paralelas umas às outras. A projeção de um ponto P_∞ dessa linha sobre o plano da imagem é dada por

$$\left(f\frac{X_0 + \lambda U}{Z_0 + \lambda W}, f\frac{Y_0 + \lambda V}{Z_0 + \lambda W} \right) .$$

À medida que $\lambda \rightarrow \infty$ ou $\lambda \rightarrow -\infty$, isso se torna $p_\infty = (fU/W, fV/W)$ se $W \neq 0$. Isso significa que duas linhas paralelas deixando pontos diferentes no espaço vão convergir na imagem — para grandes λ,

os pontos da imagem são praticamente os mesmos, qualquer que seja o valor de (X_0, Y_0, Z_0) (de novo, pense nos trilhos da ferrovia, Figura 24.1). Chamamos p_∞ o **ponto de fuga** associado com a família de linhas retas com direção (U, V, W). Linhas com a mesma direção compartilham o mesmo ponto de fuga.

24.1.2 Sistemas de lentes

A desvantagem da câmara escura é que precisamos de um furo pequeno para manter a imagem em foco. Mas, quanto menor o furo, menos fótons passam, ou seja, a imagem ficará escura. Podemos reunir mais fótons mantendo o orifício aberto por mais tempo, mas então obteremos ***perda de foco pelo movimento*** — objetos na cena que se movem aparecerão borrados porque enviam fótons para vários locais no plano da imagem. Se não pudermos manter o orifício aberto por mais tempo, podemos tentar torná-lo maior. Entrará mais luz, mas a luz de um pequeno pedaço de objeto na cena será agora distribuída por um trecho no plano da imagem, causando uma imagem desfocada.

Os olhos dos vertebrados e as câmeras modernas utilizam um sistema de lentes para reunir luz suficiente, enquanto mantêm a imagem em foco. Uma abertura grande é coberta com uma lente que focaliza a luz de objetos desde próximos até no plano da imagem. No entanto, sistemas de lentes têm **profundidade de campo** limitada: podem focalizar a luz apenas a partir de pontos que se encontram dentro de uma faixa de profundidade (centralizado em torno de um **plano focal**). Os objetos fora dessa faixa estarão fora de foco na imagem. Para mover o plano focal, as lentes do olho podem mudar de forma (Figura 24.3); em uma câmera, as lentes se movem para a frente e para trás.

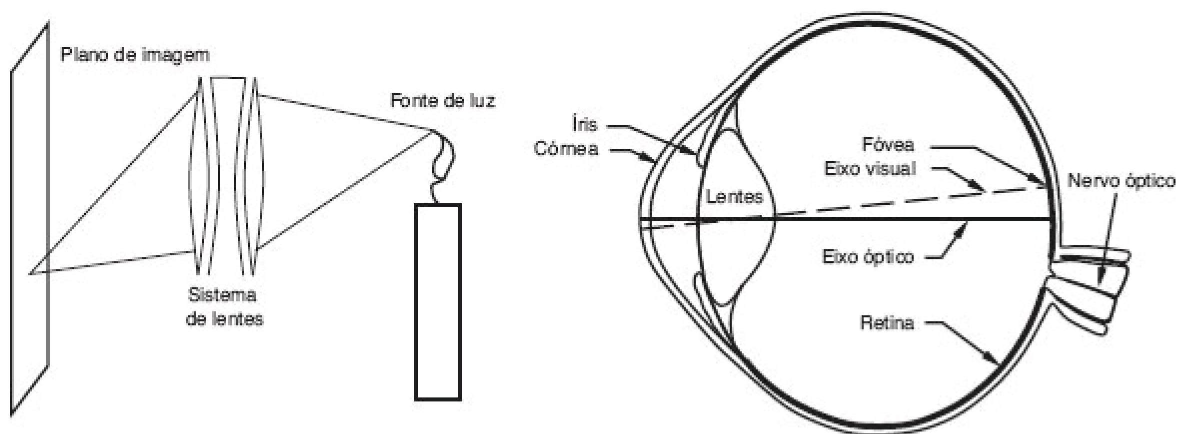


Figura 24.3 As lentes coletam luz deixando um ponto de cena em uma série de direções e orientam tudo para chegar a um único ponto no plano de imagem. O foco funciona por pontos que ficam perto de um plano focal no espaço; outros pontos não serão focados corretamente. Em câmeras, o elemento do sistema de lente move-se para mudar o plano focal, enquanto no olho a forma das lentes é alterada por músculos especializados.

24.1.3 Projeção ortográfica representada em escala

Efeitos de perspectiva nem sempre são pronunciados. Por exemplo, as manchas em um leopardo

distante podem parecer pequenas porque o leopardo está longe, mas dois pontos que estão próximos uns dos outros terão aproximadamente o mesmo tamanho. Isso ocorre porque a diferença em distância até os pontos é pequena em comparação à distância até eles, e assim podemos simplificar o modelo de projeção. O modelo apropriado é **projeção ortográfica em escala**. A ideia é a seguinte: se a profundidade Z de pontos sobre o objeto varia dentro de algum intervalo de $Z_0 \pm \Delta Z$, com $\Delta Z \lll Z_0$, o fator escalar de perspectiva f/Z pode ser aproximado por uma constante $s = f/Z_0$. As equações para a projeção de as coordenadas da cena (X, Y, Z) para o plano da imagem tornam-se $x = sX$ e $y = sY$. A projeção ortográfica em escala é uma aproximação que é válida apenas para as partes da cena com pouca variação de profundidade interna. Por exemplo, a projeção ortográfica em escala pode ser um bom modelo para as características da parte frontal de um prédio distante.

24.1.4 Luz e sombras

O brilho de um pixel na imagem é uma função do brilho do trecho da superfície na cena que se estende até o pixel. Vamos supor um modelo linear (as câmeras atuais não têm linearidades de claro e escuro nas extremidades, mas são lineares ao meio). O brilho da imagem dá uma forte, embora ambígua, sugestão da forma de um objeto e, consequentemente, sua identidade. As pessoas geralmente são capazes de distinguir as três principais causas de variação de brilho e, por engenharia reversa, das propriedades do objeto. A primeira causa é a **intensidade geral** da luz. Mesmo que um objeto branco na sombra possa ser menos brilhante do que um objeto preto na luz solar direta, o olho pode distinguir o brilho relativamente bem e perceber o objeto branco como branco. Segundo, pontos diferentes na cena podem mais ou menos **refletir** a luz. Normalmente, o resultado é que as pessoas percebem esses pontos como claros ou escuros e, assim, veem a textura ou as marcas sobre o objeto. Em terceiro lugar, trechos da superfície de frente para a luz são mais brilhantes do que trechos na superfície com inclinação para longe da luz, um efeito conhecido como **sombreamento**. Normalmente, as pessoas podem dizer que esse sombreamento vem da geometria do objeto, mas às vezes a obtenção do sombreamento e as marcações se misturam. Por exemplo, uma faixa de maquiagem escura em um osso da face, muitas vezes, parece um efeito de sombreamento, fazendo o rosto parecer mais magro.

A maioria das superfícies reflete a luz por um processo **de reflexão difusa**. A reflexão difusa dispersa a luz uniformemente entre as direções que deixam uma superfície, de modo que o brilho de uma superfície difusa não depende da direção de visualização. A maioria dos tecidos, tintas, superfícies de madeira ásperas, vegetação e pedra bruta é difusa. Os espelhos não são difusos porque o que você vê depende da direção em que olha para o espelho. O comportamento de um espelho perfeito é conhecido como **reflexão especular**. Algumas superfícies — tais como metal escovado, plástico ou chão úmido — mostram pequenos trechos onde ocorreu reflexão especular, chamados **especularidades**. Eles são fáceis de identificar porque são pequenos e brilhantes (Figura 24.4). Para quase todos os fins, é suficiente modelar todas as superfícies como sendo difusas com especularidades.



Figura 24.4 Variedade de efeitos de iluminação. Há especularidades na colher de metal e no leite. A superfície difusa brilhante é brilhante porque está de frente para a direção da luz. A superfície difusa escura é escura porque é tangente à direção da iluminação. As sombras aparecem em pontos da superfície que não podem ver a fonte de luz. Foto por Mike Linksvayer (mlinksva on flickr).

A principal fonte de iluminação exterior é o Sol, cujos raios viajam em paralelo uns com os outros. Modelamos esse comportamento como uma **fonte de ponto de luz distante**. Esse é o modelo mais importante de iluminação, e é bastante eficaz para cenas internas, assim como para cenas externas. O montante de luz recolhida por um trecho da superfície nesse modelo depende do ângulo entre a direção da iluminação e a normal à superfície.

Um trecho de superfície difusa iluminado por uma fonte de luz de ponto distante vai refletir alguma fração da luz que recolheu; essa fração é chamada de **albedo difuso**. O papel branco e a neve têm albedo elevado, cerca de 0,90, ao passo que veludo preto liso e carvão têm albedo baixo, cerca de 0,05 (o que significa que 95% da luz que entra é absorvida dentro das fibras de veludo ou dos poros de carvão). A **lei dos cossenos de Lambert** estabelece que o brilho de um trecho difuso é dado por

$$I = \rho I_0 \cos\theta,$$

onde ρ é o albedo difuso, I_0 é a intensidade da fonte de luz e θ é o ângulo entre a direção da fonte de luz e a normal da superfície (veja a Figura 24.5). A lei de Lampert prevê que os pixels da imagem brilhante vêm de trechos da superfície que confrontam a luz diretamente, e pixels escuros vêm dos trechos que veem a luz apenas tangencialmente, para que o sombreamento sobre uma superfície forneça alguma informação de forma. Vamos explorar essa pista na Seção 24.4.5. Se a superfície não for alcançada pela fonte de luz, estará na **sombra**. As sombras são muito raramente um preto uniforme porque a superfície sombreada recebe alguma luz de outras fontes. Ao ar livre, a fonte mais importante é o céu, que é bastante brilhante. Dentro, a luz refletida de outras superfícies ilumina trechos sombreados. Essas **inter-reflexões** podem ter efeito significativo sobre o brilho de outras

superfícies também. Esses efeitos são, por vezes, modelados pela adição de um termo de **iluminação do ambiente** constante com a intensidade prevista.

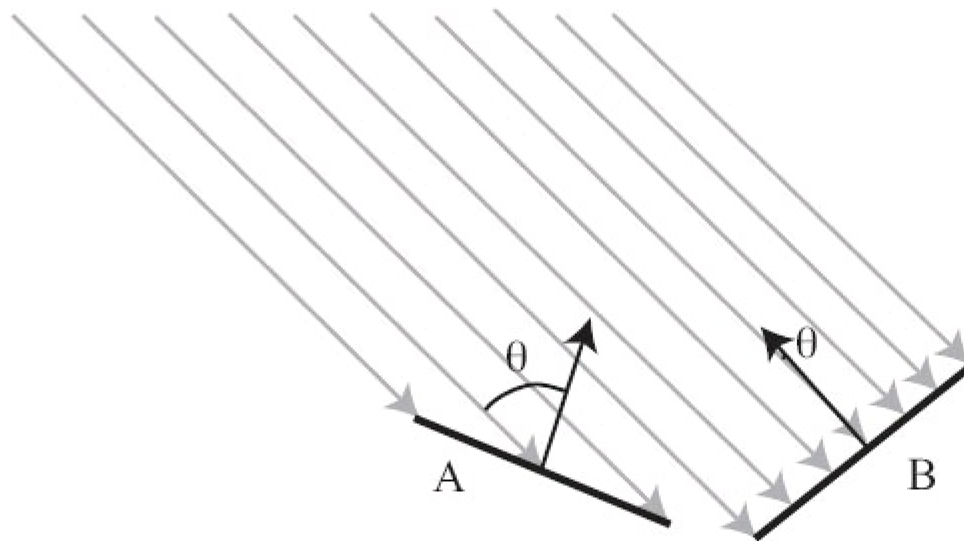


Figura 24.5 Dois trechos da superfície são iluminados por uma fonte pontual distante, cujos raios são mostrados como setas cinza. O trecho A é inclinado para longe da fonte (θ está perto de 90º) e recolhe menos energia porque corta menos raios de luz por unidade de área de superfície. O trecho B, em frente da fonte (θ está perto de 0º) recolhe mais energia.

24.1.5 Cor

A fruta é uma sedução que uma árvore oferece aos animais para transportar suas sementes ao redor. As árvores evoluíram para ter frutas que ficam vermelhas ou amarela quando maduras, e os animais evoluíram para detectar essas mudanças de cores. A luz que chega ao olho tem diferentes quantidades de energia em comprimentos de onda diferentes; isso pode ser representado por uma função de densidade de energia espectral. Os olhos humanos respondem à luz na região de comprimento de onda de 380-750 nm, com três diferentes tipos de células receptoras da cor, que têm pico de receptividade em 420 mm (azul), 540 nm (verde) e 570 nm (vermelho). O olho humano pode capturar apenas uma pequena fração da função de densidade de energia espectral total, mas é o suficiente para informar quando a fruta está madura.

O **princípio da tricromacia** afirma que, para qualquer densidade de energia espectral, não importa o quão complicado, é possível construir outra densidade de energia espectral constituída por uma mistura de apenas três cores — geralmente vermelho, verde e azul — de tal forma que um ser humano não possa dizer a diferença entre os dois. Isso significa que os nossos televisores e monitores de computador podem conviver com apenas os três elementos de cor, vermelho/verde/azul (ou R/G/B). Isso também torna mais fáceis nossos algoritmos de visão computacional. Cada superfície pode ser modelada com três albedos diferentes para R/G/B.

Da mesma forma, cada fonte de luz pode ser modelada com três intensidades diferentes R/G/B. Em seguida, aplicamos a lei dos cossenos de Lambert para cada uma para obter três valores de pixel R/G/B. Esse modelo prevê, corretamente, que a mesma superfície vai produzir diferentes trechos de imagem colorida com luzes de cores diferentes. De fato, os observadores humanos são muito bons em

ignorar os efeitos de luz com cores diferentes e são capazes de estimar a cor da superfície sob a luz branca, efeito conhecido como **constância de cores**. Atualmente estão disponíveis algoritmos de constância de cores bastante precisos; na função de sua câmera "balanço automático do branco", aparecem versões simples. Observe que, se quisermos construir uma câmera para zagaias, precisaremos de 12 cores diferentes de pixels, correspondentes aos 12 tipos de receptores de cor do crustáceo.

24.2 OPERAÇÕES INICIAIS DE PROCESSAMENTO DE IMAGENS

Vimos como a luz reflete objetos da cena para formar uma imagem que consiste em, digamos, cinco milhões de pixels de três bytes. Com todos os sensores haverá ruído na imagem e, em qualquer caso, existe grande quantidade de dados para lidar com eles. Então, como vamos começar a analisar esses dados?

Nesta seção vamos estudar três operações úteis de processamento de imagem: detecção de arestas, análise de textura e cálculo de fluxo óptico.

Essas operações são chamadas "iniciais" ou de "baixo nível" porque são as primeiras de uma série de operações. As operações iniciais de visão são caracterizadas por sua natureza local (elas podem ser executadas em uma parte da imagem sem considerar nada além de alguns pixels de distância) e por sua falta de conhecimento: podemos suavizar imagens e detectar arestas sem ter qualquer ideia de quais são os objetos nas imagens. Isso torna as operações de baixo nível boas candidatas para implementação em hardware paralelo — seja em unidade de processamento gráfico (UPG) ou um olho. Em seguida, examinaremos uma operação de nível intermediário, a segmentação da imagem em regiões.

24.2.1 Detecção de arestas

As **arestas** são linhas retas ou curvas no plano da imagem, ao longo das quais existe uma mudança "significativa" no brilho da imagem. A meta da detecção de arestas é abstrair a imagem da imagem confusa de vários megabytes e orientá-la para uma representação mais compacta e abstrata, como na Figura 24.6. A motivação é que os contornos de arestas na imagem correspondem a contornos importantes da cena. Na figura, temos três exemplos de descontinuidade da profundidade, identificados por 1; duas descontinuidades de orientação da superfície, identificadas por 2; uma descontinuidade de refletância, identificada por 3; e uma descontinuidade de iluminação (sombra), identificada por 4. A detecção de arestas se preocupa apenas com a imagem e, desse modo, não faz distinção entre esses tipos diferentes de descontinuidades na cena, mas o processamento posterior os distinguirá.

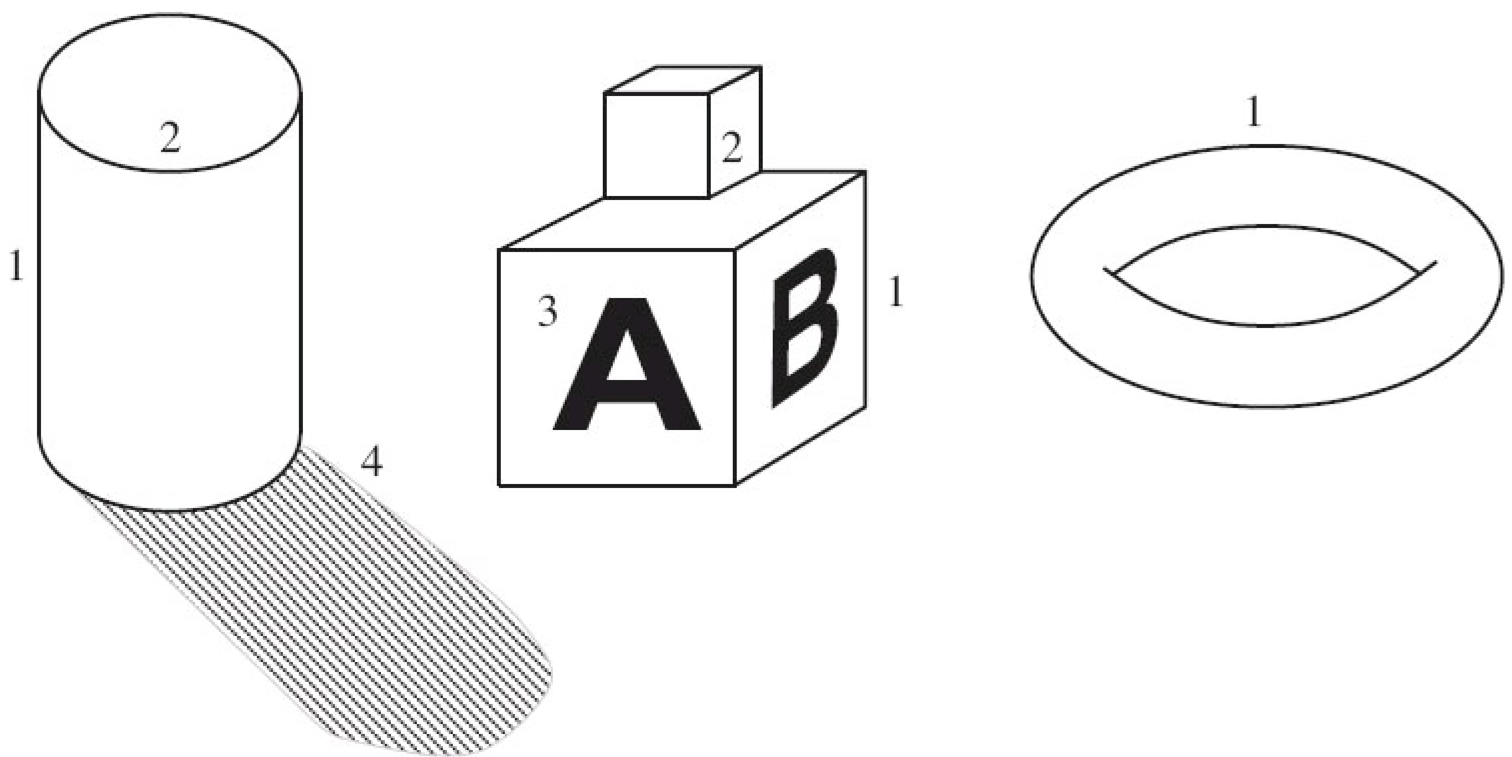
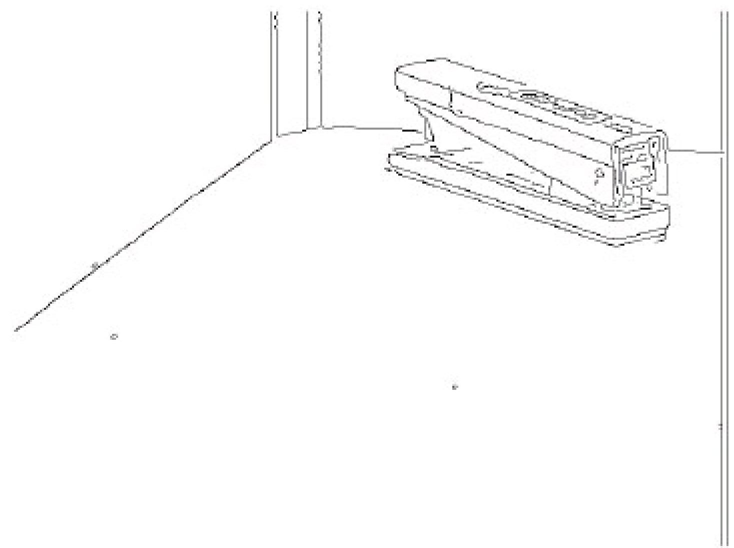


Figura 24.6 Diferentes tipos de arestas: (1) descontinuidades de profundidade; (2) descontinuidades de orientação da superfície; (3) descontinuidades de refletância; (4) descontinuidades de iluminação (sombras).

A Figura 24.7(a) mostra a imagem de uma cena contendo um grampeador que repousa sobre uma escrivaninha e (b) mostra a saída de um algoritmo de detecção de arestas sobre essa imagem. Como podemos ver, existe diferença entre a saída e um desenho a traço ideal. Existem lacunas onde não aparecem arestas, e há arestas com "ruído" que não correspondem a nada de significativo na cena. Estágios posteriores de processamento deverão corrigir esses erros.



(a)



(b)

Figura 24.7 (a) Fotografia de um grampeador. (b) Arestas calculadas a partir de (a).

Como podemos detectar arestas em uma imagem? Considerando o perfil de brilho da imagem ao longo de uma seção transversal unidimensional perpendicular a uma das arestas — por exemplo, a que está entre a aresta esquerda da escrivaninha e a parede. Parece algo como o que é mostrado na

Figura 24.8 (parte superior).

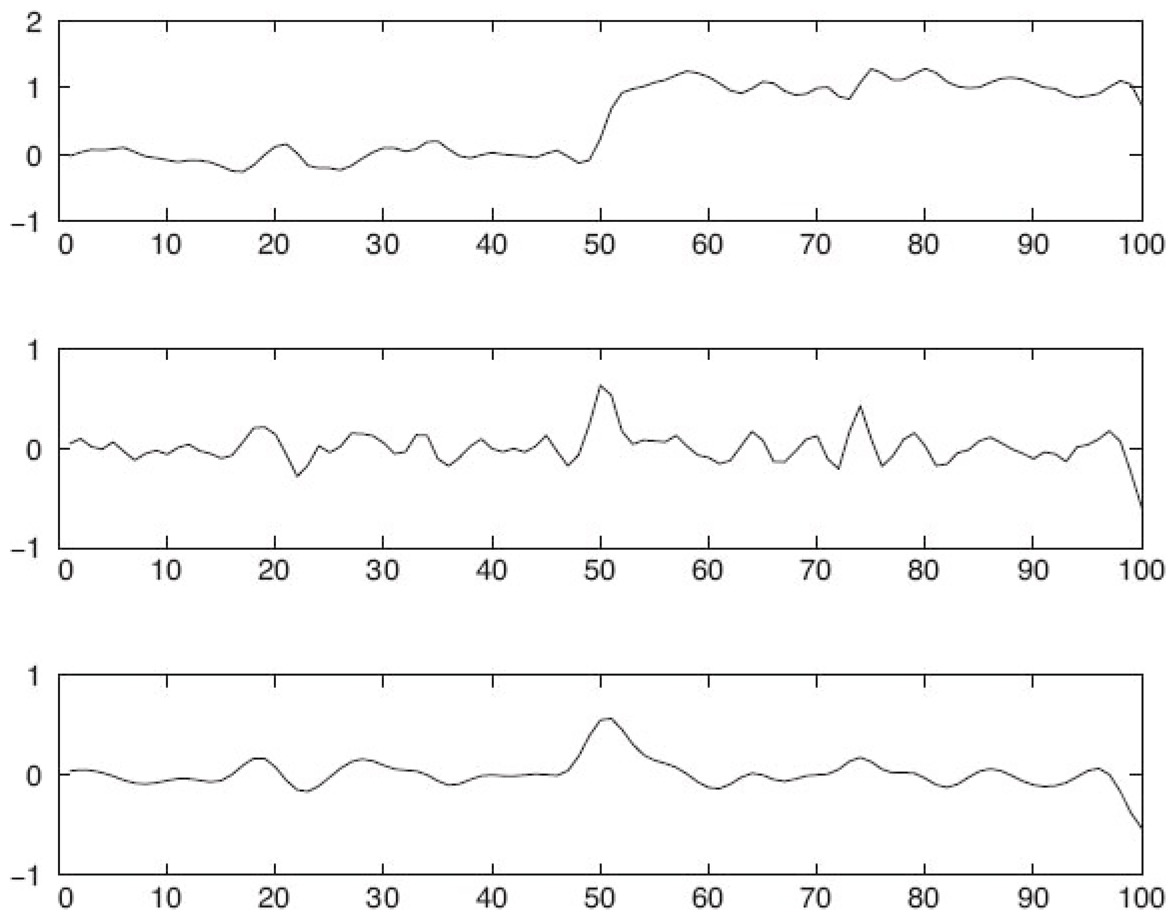


Figura 24.8 Parte superior: perfil de intensidade de $I(x)$ ao longo de uma seção unidimensional através de uma aresta em $x = 50$. Parte do meio: a derivada de intensidade, $I'(x)$. Grandes valores dessa função correspondem às arestas, mas a função é ruidosa. Parte inferior: a derivada de uma versão suavizada da intensidade, $(I * G_\sigma)'$, que pode ser calculada em uma etapa como a convolução $I * G'_\sigma$ A aresta candidata ruidosa em $x = 75$ desapareceu.

As arestas correspondem aos locais em imagens onde o brilho sofre uma mudança brusca; assim, uma ideia ingênua seria diferenciar a imagem e olhar para os lugares onde a magnitude do derivado $I'(x)$ é grande. Isso quase funciona. Na Figura 24.8 (meio), vemos que de fato há um pico em $x = 50$, mas há também picos subsidiários em outros locais (por exemplo, $x = 75$). Isso surge por causa da presença de ruído na imagem. Se suavizarmos a primeira imagem, os picos espúrios serão diminuídos, como vemos no fundo da figura.

A medição do brilho de um pixel em uma câmera CCD é baseada em um processo físico que envolve a absorção de fótons e a liberação de elétrons; há flutuações estatísticas da medição — ruído. O ruído pode ser modelado com uma distribuição de probabilidade gaussiana, com cada pixel independente dos outros. Uma maneira de suavizar uma imagem é atribuir a cada pixel a média de seus vizinhos. Isso tende a cancelar valores extremos. Mas devemos considerar quantos vizinhos — um pixel distante ou dois ou mais? Uma boa resposta é uma média ponderada que dê o maior peso aos pixels mais próximos, diminuindo gradualmente o peso dos pixels mais distantes. O **filtro gaussiano** faz exatamente isso (os usuários do Photoshop reconhecem isso como operação de *desfoque gaussiano*). Lembre-se de que a função gaussiana com desvio-padrão σ e média 0 é

$$N_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2} \qquad \text{em uma dimensão ou}$$
$$N_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \qquad \text{em duas dimensões.}$$

A aplicação do filtro gaussiano substitui a intensidade $I(x_0, y_0)$ com a soma, sobre todos os pixels (x, y) de $I(x, y)\, N_\sigma(d)$, onde d é a distância de (x_0, y_0) a (x, y). Esse tipo de soma ponderada é tão comum que há um nome especial e uma notação para ela. Dizemos que a função h é a **convolução** de duas funções f e g (indicada $f * g$) se tivermos

$$h(x) = (f * g)(x) = \sum_{u=-\infty}^{+\infty} f(u)\, g(x-u) \qquad \text{em uma dimensão ou}$$

$$h(x,y) = (f * g)(x,y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u,v)\, g(x-u, y-v) \qquad \text{em duas.}$$

Assim, a função de alisamento é atingida através da convolução da imagem com a gaussiana, $I * N\sigma$. Um σ de 1 pixel é suficiente para suavizar uma pequena quantidade de ruído, enquanto 2 pixels vão suavizar uma quantidade maior, mas com a perda de alguns detalhes. Como a influência da gaussiana desaparece rapidamente à distância, podemos substituir $\pm\infty$ na soma por $\pm 3\sigma$.

Podemos otimizar o cálculo combinando alisamento e encontro da aresta em uma única operação. É um teorema que, para quaisquer funções f e g, a derivada da convolução, $(f * g)'$, é igual à convolução com a derivada, $f * (g')$. Então, em vez de alisar a imagem e, em seguida, diferenciar, podemos apenas convolver a imagem com a derivada da função de alisamento, $N'\sigma$. Em seguida, marcamos como arestas os picos que na resposta estão acima de algum limite.

Há uma generalização natural desse algoritmo de seções transversais unidimensionais a imagens bidimensionais em geral. Em duas dimensões, as arestas podem estar em qualquer ângulo θ. Considerando-se o brilho da imagem como uma função escalar das variáveis x, y, seu gradiente é um vetor

$$\nabla I = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix} = \begin{pmatrix} I_x \\ I_y \end{pmatrix} .$$

As arestas correspondem aos locais nas imagens onde o brilho sofre uma mudança brusca e, assim, a magnitude do gradiente, $\|\nabla I\|$, deve ser grande em um ponto da aresta. A direção do gradiente é de interesse independente.

$$\frac{\nabla I}{\|\nabla I\|} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} .$$

Isso nos dá um $\theta = \theta(x, y)$ em cada pixel, que define a **orientação** da aresta naquele pixel.

Como em uma dimensão, para formar o gradiente não calculamos ∇I, mas $\nabla (I * N_\sigma)$, o gradiente

após o alisamento da imagem envolvendo-a com uma gaussiana. E, novamente, o atalho é que isso é equivalente a envolver a imagem com as derivadas parciais de uma gaussiana. Uma vez que calculamos o gradiente, podemos obter arestas ao encontrar pontos de arestas e ligá-los. Para saber se um ponto é um ponto da aresta, temos de olhar para outros pontos de uma pequena distância para a frente e para trás ao longo da direção do gradiente. Se a magnitude do gradiente em um desses pontos for maior, poderemos obter um melhor ponto da aresta deslocando a curva da aresta muito ligeiramente. Além disso, se a magnitude do gradiente for muito pequena, o ponto não pode ser um ponto da aresta. Então, em um ponto da aresta, a magnitude do gradiente é um máximo local ao longo da direção do gradiente, estando a magnitude do gradiente acima de um limite adequado.

Uma vez que tenhamos marcado os pixels da aresta por esse algoritmo, a próxima etapa é ligar esses pixels que pertencem às mesmas curvas da aresta. Isso pode ser feito assumindo que quaisquer dois pixels da aresta adjacentes com orientações consistentes devem pertencer à mesma curva da aresta.

24.2.2 Textura

Na linguagem cotidiana, **textura** é a sensação visual de uma superfície — o que você vê evoca o que a superfície poderia sentir se você a tocasse ("textura" tem a mesma raiz que "têxtil"). Em visão computacional, textura refere-se a um padrão que se repete espacialmente em uma superfície que pode ser percebida visualmente. Os exemplos incluem o padrão de janelas em um edifício, os pontos em um suéter, as manchas em um leopardo, folhas de grama em um gramado, cristais de rocha em uma praia e as pessoas em um estádio. Às vezes, o arranjo é bastante periódico, como os pontos em um suéter; em outros casos, tais como cristais de rocha em uma praia, a regularidade é apenas estatística.

Considerando que o brilho é uma propriedade de pixels individuais, o conceito de textura faz sentido apenas para um trecho multipixel. Dado tal trecho, poderíamos calcular a orientação em cada pixel e caracterizar o trecho por um histograma de orientações. A textura de tijolos em uma parede teria dois picos no histograma (um vertical e um horizontal), enquanto a textura das manchas na pele de um leopardo teria uma distribuição mais uniforme de orientações.

A Figura 24.9 mostra que as orientações são em grande parte invariantes às alterações na iluminação. Isso torna a textura uma pista importante para o reconhecimento do objeto porque outras pistas, tais como arestas, podem produzir resultados diferentes em diferentes condições de iluminação.


(a)

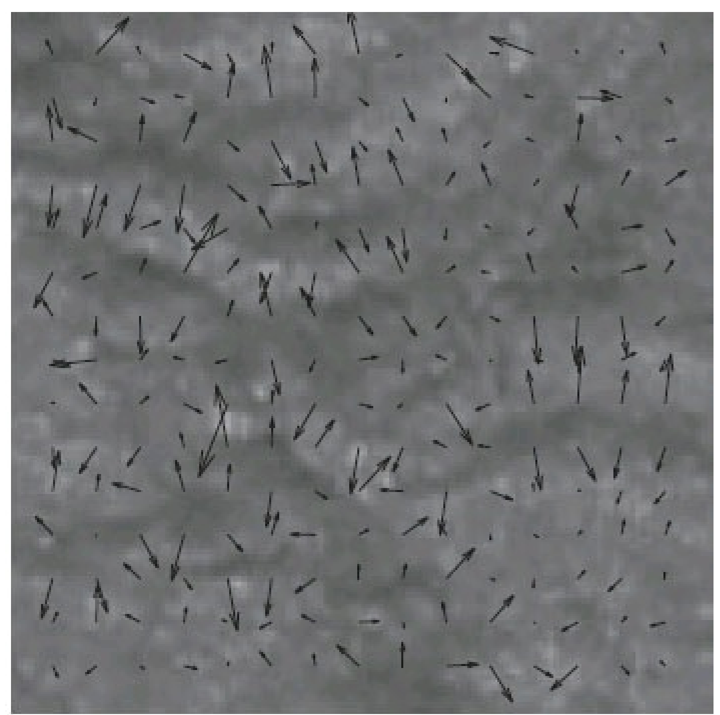

(b)

Figura 24.9 Duas imagens da mesma textura de papel de arroz amassado, com níveis de iluminação diferentes. O campo vetorial gradiente (em cada oitavo pixel) é traçado em cima de cada um. Observe que, à medida que a luz fica mais escura, todos os vetores gradientes ficam mais curtos. Os vetores não giram, de modo que a orientação do gradiente não muda.

Em imagens de objetos texturizados, a detecção de arestas não funciona tão bem como em objetos lisos. Isso ocorre porque as arestas mais importantes podem ser perdidas entre os elementos de textura.

Literalmente, podemos perder o tigre para as listras. A solução é olhar para as diferenças de propriedades de textura, da mesma forma que olhamos para as diferenças de luminosidade. Um trecho em um tigre e um trecho no fundo de um gramado terá histogramas com orientação muito diferentes, permitindo-nos encontrar a curva de fronteira entre eles.

24.2.3 Fluxo óptico

Em seguida, vamos considerar o que acontece quando temos uma sequência de vídeo, em vez de apenas uma única imagem estática. Quando um objeto no vídeo está se movendo ou quando a câmera se move em relação a um objeto, o movimento aparente resultante na imagem é chamado de **fluxo óptico**. O fluxo óptico descreve a direção e a velocidade do movimento de atributos dentro da *imagem* — o fluxo óptico do vídeo de uma corrida de carro seria medido em pixels por segundo, e não em quilômetros por hora. O fluxo óptico codifica informação útil sobre a estrutura da cena. Por exemplo, em um vídeo de um cenário tirado de um trem em movimento, os objetos distantes têm movimento aparentemente mais lento do que os objetos próximos; assim, a taxa de movimento aparente pode nos informar algo sobre a distância. O fluxo óptico também nos permite reconhecer ações. Na Figura 24.10(a) e (b), mostramos dois quadros de um vídeo de um jogador de tênis. Em (c) apresentamos os vetores de fluxo óptico calculado a partir dessas imagens, mostrando que a raquete

e a perna da frente são mais rápidas.



Figura 24.10 Dois quadros de uma sequência de vídeo. À direita está o campo de fluxo óptico correspondente ao deslocamento de um quadro para o outro. Observe como o movimento da raquete de tênis e da perna da frente é capturado pelas direções das setas. (Cortesia de Thomas Brox.)

O campo vetorial de fluxo óptico pode ser representado em qualquer ponto (x, y) por seus componentes $v_x(x, y)$ na direção x e $v_y(x, y)$ na direção y. Para medir o fluxo óptico é preciso encontrar pontos correspondentes entre um período de tempo e o próximo. Uma técnica simplória baseia-se no fato de que pequenas áreas de imagem em torno dos pontos correspondentes têm os mesmos padrões de intensidade. Considere um bloco de pixels centralizado no pixel p, (x_0, y_0), no momento t_0. Esse bloco de pixels deve ser comparado com blocos de pixels centralizados em vários pixels candidatos em $(x_0 + D_x, y_0 + D_y)$ no tempo $t_0 + D_t$. Uma medida possível de similaridade é a **soma dos quadrados das diferenças** (SQD):

$$\mathrm{SSD}(D_x, D_y) = \sum_{(x,y)} (I(x,y,t) - I(x + D_x, y + D_y, t + D_t))^2 \,.$$

Aqui, (x, y) varia ao longo dos pixels no bloco centralizado em (x_0, y_0). Encontramos o (D_x, D_y) que minimiza a SQD. O fluxo óptico em (x_0, y_0) é então $(v_x, v_y) = (D_x/D_t, D_y/D_t)$. Observe que, para que isso funcione, é preciso haver alguma textura ou variação na cena. Se alguém estiver procurando uma parede branca uniforme, a SQD vai ser quase a mesmo para as partidas candidatas diferentes, e o algoritmo é reduzido para fazer uma suposição cega. Os algoritmos com melhor desempenho para medir o fluxo óptico dependem de uma variedade de restrições adicionais quando a cena é apenas parcialmente texturizada.

24.2.4 Segmentação de imagens

A **segmentação** é o processo de desmembrar uma imagem em regiões de pixels semelhantes. Cada pixel de imagem pode ser associado a certas propriedades visuais, como brilho, cor e textura. Dentro de um objeto ou em uma única parte de um objeto, esses atributos variam relativamente pouco; por

outro lado, ao longo de um limite entre objetos, em geral existe uma grande mudança em um ou outro desses atributos. Existem duas abordagens para a segmentação: uma com foco em detectar os limites dessas regiões e a outra na detecção das próprias regiões (Figura 24.11).

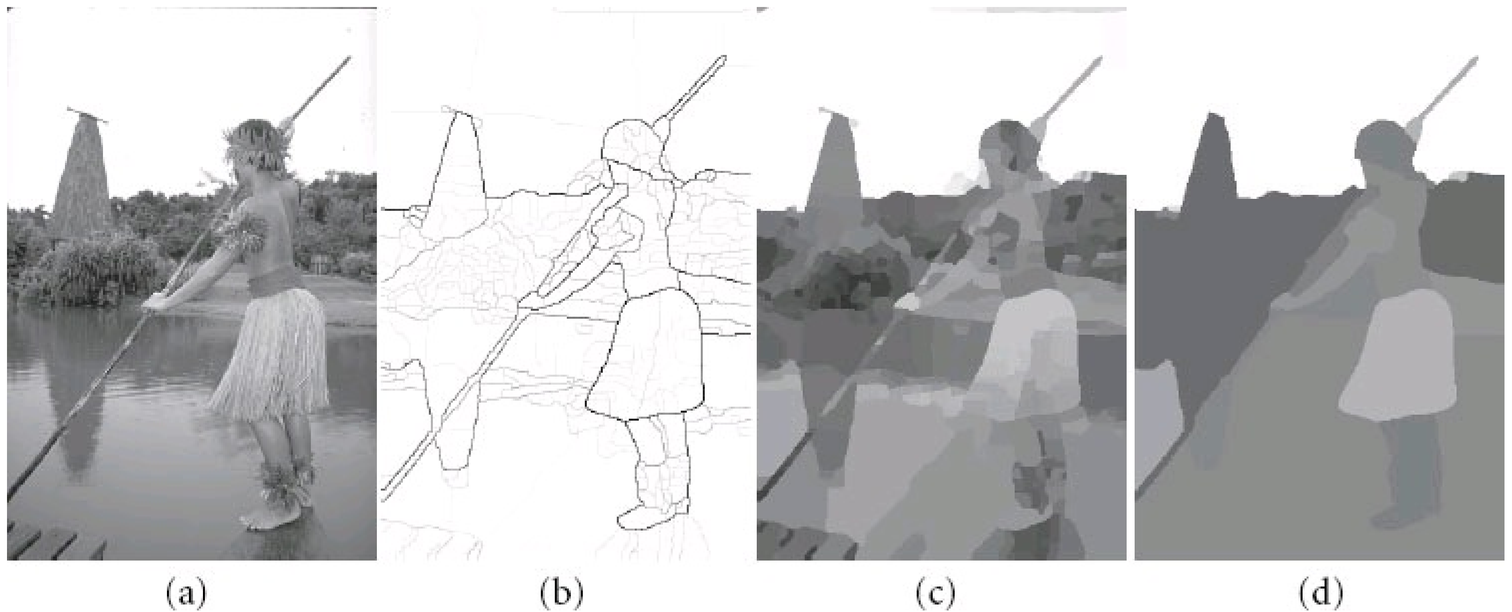


Figura 24.11 (a) Imagem original. (b) Contornos de fronteira onde, quanto maior for o valor de P_b, mais escuro o contorno. (c) Segmentação em regiões, que correspondem a uma partição fina da imagem. As regiões são apresentadas em suas cores médias. (d) Segmentação em regiões, correspondendo a uma partição grosseira da imagem, resultando em menos regiões. (Cortesia de Pablo Arbelaez, Michael Maire, Charles Fowlkes e Jitendra Malik.)

Uma curva de contorno passando por um pixel (x, y) terá uma orientação θ; assim, uma forma de formalizar o problema de detecção de curvas de contorno é como um problema de classificação de aprendizado de máquina. Com base em características de uma vizinhança local, queremos calcular a probabilidade $P_b(x, y, \theta)$ que existe realmente uma curva de contorno naquele pixel ao longo dessa orientação. Considere um disco circular centralizado em (x, y), subdividido em dois meios discos por um diâmetro orientado em θ. Se houver um limite em (x, y, θ) pode-se esperar que os dois meios discos sejam significativamente diferentes em seu brilho, cor e textura. Martin, Fowlkes e Malik (2004) utilizaram recursos com base nas diferenças de histogramas de cor, brilho e textura de valores medidos nesses dois meios discos e depois treinaram um classificador. Para isso, eles usaram um conjunto de dados de imagens naturais, onde os seres humanos marcaram os limites da "área verdadeira", e o objetivo do classificador era marcar exatamente aqueles limites marcados pelos seres humanos, e não outros.

Os limites detectados por essa técnica vêm a ser significativamente melhores do que aqueles encontrados utilizando a técnica de detecção de arestas simples descrito anteriormente. Mas ainda há duas limitações. (1) O limite de pixels formado pelo limiar $P_b(x, y, \theta)$ não é garantido de formar curvas fechadas; assim, essa abordagem não resgata regiões. (2) A tomada de decisão utiliza apenas o contexto local e não usa restrições de consistência global.

A abordagem alternativa é baseada na tentativa de "agrupar" os pixels em regiões com base em seu brilho, cor e textura. Shi e Malik (2000) configuraram isso como um particionamento gráfico. Os

nós do grafo correspondem aos pixels, e as arestas, às conexões entre os pixels. O peso W_{ij} na aresta conectando um par de pixels i e j é baseado no quão semelhante são os dois pixels em brilho, cor, textura etc. São então encontradas partições que minimizam um critério de *corte normalizado*.

Grosso modo, o critério para particionar o grafo é minimizar a soma dos pesos das conexões entre os grupos de pixels e maximizar a soma dos pesos de conexões dentro dos grupos.

Não se pode esperar da segmentação baseada puramente em atributos locais de baixo nível, como brilho e cor, que entregue os limites finais corretos de todos os objetos na cena. Para descobrir de modo confiável limites associados a objetos, também devemos incorporar o conhecimento de alto nível dos tipos de objetos que podemos esperar encontrar em uma cena. Representar esse conhecimento é um tema de pesquisa ativa. Uma estratégia popular é produzir excesso de segmentação de uma imagem, contendo centenas de regiões homogêneas que são conhecidas como **superpixels**. De lá, os algoritmos baseados em conhecimento podem assumir; acharão mais fácil lidar com centenas de superpixels, em vez de milhões de pixels em estado natural. O assunto da próxima seção é como explorar o conhecimento de objetos de alto nível.

24.3 RECONHECIMENTO DE OBJETO POR APARÊNCIA

A **aparência** é uma abreviação para o que um objeto tende a parecer. Algumas categorias de objetos — por exemplo, bolas de beisebol — variam muito pouco na aparência; todos os objetos na categoria parecem o mesmo na maioria das circunstâncias. Nesse caso, podemos calcular um conjunto de recursos descrevendo cada classe de imagens com probabilidade de conter o objeto e, em seguida, testá-lo com um classificador.

Outras categorias de objeto — por exemplo, casas ou bailarinos — variam muito. Uma casa pode ter tamanho, cor e forma diferentes, e pode parecer diferente de ângulos diferentes. Um dançarino parece diferente em cada pose ou quando as luzes do palco mudam de cores. Uma abstração útil é dizer que alguns objetos são feitos de padrões locais que tendem a se movimentar com relação uns aos outros. Podemos, então, encontrar o objeto olhando para histogramas locais de detector de respostas, que expõem se alguma parte está presente, mas suprime os detalhes de onde ela está.

O teste de cada classe de imagens com um classificador instruído é uma receita geral importante. Funciona muito bem para rostos olhando diretamente para a câmera porque, em baixa resolução e sob iluminação razoável, todos os rostos parecem bastante semelhantes. O rosto é redondo e bastante brilhante, em comparação com as órbitas dos olhos, que são escuras porque estão submersas, e a boca é um talho escuro, assim como as sobrancelhas. Grandes mudanças de iluminação podem causar algumas variações nesse padrão, mas o intervalo de variação é bastante controlável. Torna possível a detecção de posições do rosto em uma imagem que contém rostos. Outrora um desafio computacional, esse recurso é agora comum mesmo em câmeras digitais baratas.

Por enquanto, vamos considerar apenas faces nas quais o nariz é orientado verticalmente; lidaremos adiante com faces em revolução. Passamos uma janela redonda de tamanho fixo sobre a imagem, calculamos os respectivos atributos e apresentamos estes atributos para um classificador. Essa estratégia é, por vezes, chamada de **janela deslizante**. As características precisam ser robustas

com relação a sombras e a mudanças no brilho causadas por mudança na iluminação. Uma estratégia é a construção de atributos a partir de orientações do gradiente. Outra é estimar e corrigir a iluminação em cada janela de imagem. Para encontrar rostos de tamanhos diferentes, repita a varredura em relação às versões maiores ou menores da imagem. Finalmente, processamos posteriormente as respostas através de escalas e locais para produzir o conjunto final de detecções.

O pós-processamento é importante porque é improvável que tenhamos escolhido um tamanho de janela que seja exatamente o tamanho certo para um rosto (mesmo que usemos vários tamanhos). Assim, provavelmente teremos várias janelas sobrepostas, cada uma registrando a combinação com um rosto. No entanto, se utilizarmos um classificador que possa registrar potência na resposta (por exemplo, a regressão logística ou uma máquina de vetores de suporte), podemos combinar essas correspondências parciais sobrepostas em locais próximos para produzir uma correspondência única de alta qualidade. Isso nos fornece um detector de rosto que pode pesquisar sobre locais e escalas. Para pesquisar também rotações, usamos duas etapas. Treinamos um procedimento de regressão para estimar a melhor orientação de qualquer rosto presente em uma janela. Agora, para cada janela, estimamos a orientação, reorientamos a janela e testamos se um rosto vertical está presente com o nosso classificador. Tudo isso produz um sistema cuja arquitetura está esboçada na Figura 24.12.

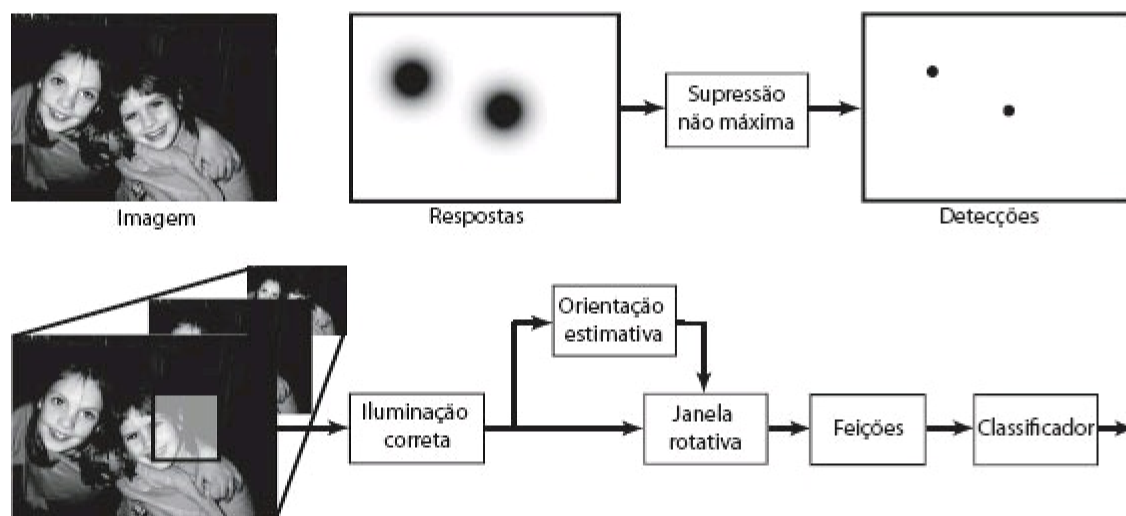


Figura 24.12 Os sistemas de detecção de rosto variam, mas a maioria segue a arquitetura ilustrada nessas duas partes aqui. No topo, vamos de imagens a respostas, em seguida aplicamos a supressão não máxima para encontrar a resposta local mais forte. As respostas são obtidas pelo processo ilustrado na parte inferior. Varremos uma janela de tamanho fixo em relação às versões maiores e menores da imagem, de forma a encontrar faces menores ou maiores, respectivamente. A iluminação na janela é corrigida e depois um mecanismo de regressão (muitas vezes, uma rede neural) prevê a orientação da face. A janela é corrigida com essa orientação e, em seguida, apresentada a um classificador. As saídas do classificador são então pós-processadas para garantir que apenas um rosto seja colocado em cada local na imagem.

Dados de treinamento são obtidos facilmente. Há vários conjuntos de dados de imagens de rosto remarcados, e janelas com rostos virados são fáceis de construir (basta girar uma janela de um conjunto de dados de treinamento). Um truque que é amplamente utilizado é o de pegar cada janela de exemplo e em seguida produzir novos exemplos, mudando a orientação da janela, o centro da janela ou a escala muito ligeiramente. Essa é uma maneira fácil de obter um conjunto maior de dados que

reflete imagens reais bastante bem; o truque geralmente melhora o desempenho significativamente. Os detectores de rosto construídos com essas linhas apresentam bom desempenho para rostos frontais (a visão lateral é a mais difícil).

24.3.1 Aparência complexa e elementos-padrão

Muitos objetos produzem padrões muito mais complexos do que os rostos porque vários efeitos podem movimentar características ao redor de uma imagem do objeto. Os efeitos incluem (Figura 24.13):

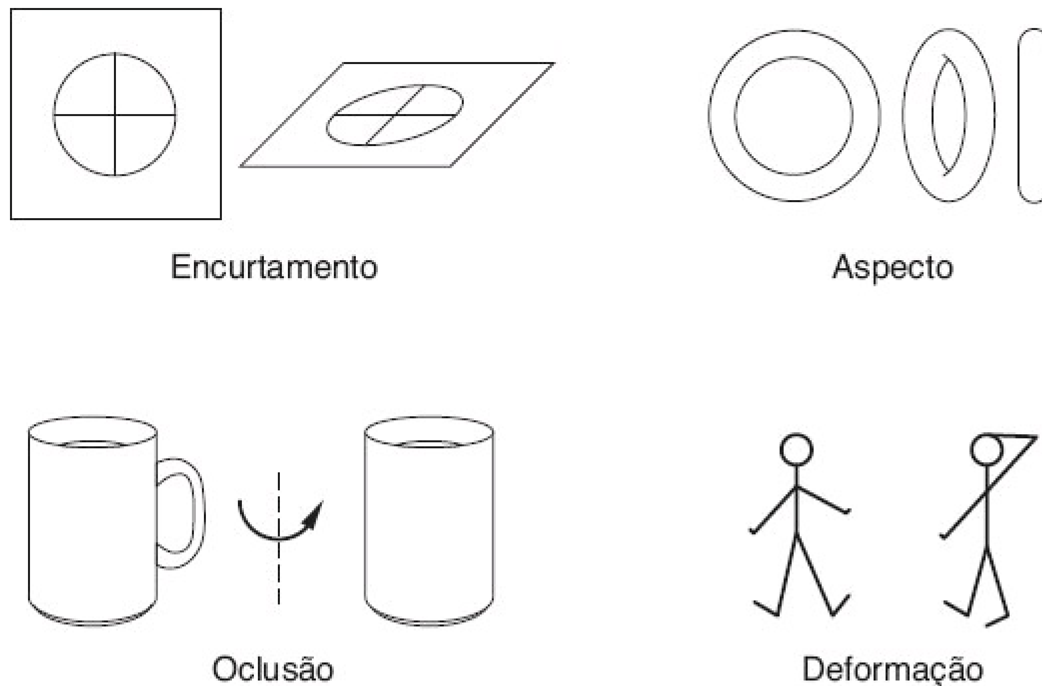


Figura 24.13 Fontes de variação de aparência. Primeiro, os elementos podem encurtar, como o trecho circular no canto superior esquerdo. Esse trecho é visto em uma inclinação e, assim, a imagem é elíptica. Segundo, objetos vistos de direções diferentes podem mudar de forma drasticamente, um fenômeno conhecido como aspecto. No canto superior direito há três aspectos diferentes de um *donut*. A oclusão faz com que a alça da caneca no canto inferior esquerdo desapareça quando a caneca é girada. Nesse caso, como o corpo e a alça pertencem à mesma caneca, temos a auto-oclusão. Finalmente, no canto inferior direito, alguns objetos podem deformar drasticamente.

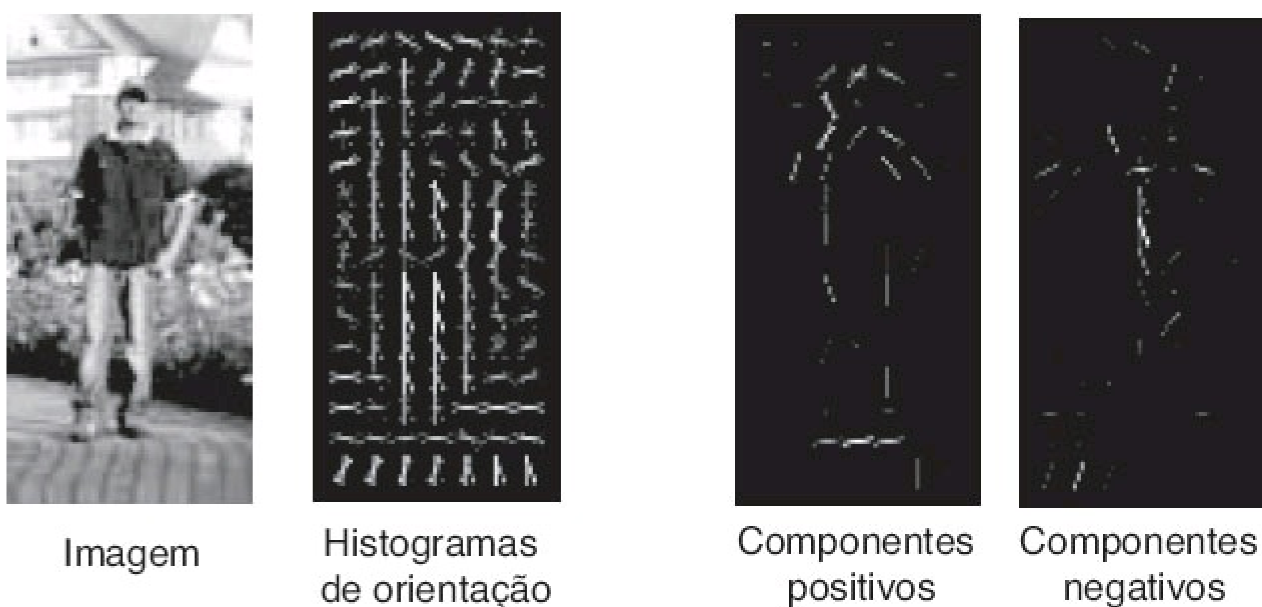


Figura 24.14 Os histogramas de orientação local são um recurso poderoso para o reconhecimento, mesmo de objetos bastante complexos. À esquerda, a imagem de um pedestre. No centro, à esquerda, histogramas de orientação local de trechos. Em seguida, aplica-se um classificador como uma máquina de vetores de suporte para encontrar os pesos de cada histograma que separam melhor os exemplos positivos de pedestres de não pedestres. Vemos que os componentes positivamente ponderados parecem com o contorno de uma pessoa. Os componentes negativos são menos claros; representam todos os padrões de não pedestres. Figura de Dalal e Triggs (2005) © IEEE.

- **Encurtamento**, que provoca um padrão visto em inclinação significativamente distorcida.
- **Aspecto**, que faz com que os objetos pareçam diferentes quando vistos de direções diferentes. Mesmo um objeto mais simples, como um *donut*, tem vários aspectos; visto de lado, parece um objeto oval achatado, mas de cima é um anel.
- **Oclusão**, onde algumas partes estão escondidas de algumas direções de visualização. Os objetos podem ocluir uns aos outros ou partes de um objeto podem ocluir outras partes, um efeito conhecido como auto-oclusão.
- **Deformação**, onde graus internos de liberdade do objeto alteram sua aparência. Por exemplo, as pessoas podem mover seus braços e pernas, gerando uma gama muito ampla de configurações diferentes do corpo.

No entanto, nossa receita de pesquisa através de localização e escala ainda pode funcionar porque alguma estrutura estará presente nas imagens produzidas pelo objeto. Por exemplo, é provável que a imagem de um carro mostre alguns dos faróis, portas, rodas, janelas e calotas, embora possam estar em arranjos um pouco diferentes em imagens diferentes. Isso sugere modelar objetos com elementos-padrão — coleções de partes. Esses elementos-padrão podem se mover em relação uns aos outros, mas, se a maioria dos elementos-padrão estiver presente no lugar certo, o objeto estará presente. Um reconhecedor de objetos é, então, uma coleção de recursos que pode informar se o padrão de elementos está presente e se eles estão no lugar certo.

A abordagem mais óbvia é a de representar a janela de imagem com um histograma de elementos-padrão que lá aparecem. Essa abordagem não funciona muito bem porque muitos padrões se confundem uns com os outros. Por exemplo, se os elementos-padrão são pixels de cor, as bandeiras

francesa, inglesa, holandesa se confundem porque têm aproximadamente os histogramas da mesma cor, embora as cores estejam dispostas de maneiras diferentes. Modificações bastante simples de histogramas rendem características muito úteis. O truque é preservar alguns detalhes espaciais na representação; por exemplo, faróis tendem a estar na frente de um carro, e as rodas tendem a estar na parte inferior. Histogramas baseados em características tiveram sucesso em ampla variedade de aplicações de reconhecimento; vamos pesquisar detecção de pedestres.

24.3.2 Detecção de pedestres com características HOG

O Banco Mundial calcula que os acidentes de carro a cada ano matem cerca de 1,2 milhão de pessoas, das quais cerca de dois terços são pedestres. Isso significa que a detecção de pedestres é um problema de aplicação importante porque os carros que podem detectar automaticamente e evitar os pedestres podem salvar muitas vidas. Os pedestres utilizam muitos tipos diferentes de roupas e aparecem em muitas configurações diferentes, mas, com resolução relativamente baixa, eles podem ter aparência característica bastante regular. Os casos mais comuns são as vistas laterais ou frontais de uma caminhada. Nesses casos, vemos a forma de um "pirulito" — o tronco é maior do que as pernas, que estão juntas na fase de posição de início da caminhada — ou uma forma de "tesoura" — em que as pernas estão balançando na caminhada. Esperamos ver alguma evidência de braços e pernas, e a curva em torno dos ombros e a cabeça também tende a visível e bastante distinta. Isso significa que, com uma construção de característica cuidadosa, podemos construir um detector de pedestres útil com janela móvel.

Nem sempre há um contraste forte entre o pedestre e o fundo, por isso é melhor usar orientações que arestas para representar a janela de imagem. Os pedestres podem mover seus braços e pernas ao redor; por isso, devemos usar um histograma para suprimir alguns detalhes espaciais. Quebramos a janela em células, o que pode sobrepor, e construímos um histograma de orientação em cada célula. Fazer isso vai produzir um atributo que pode informar se a curva da cabeça e dos ombros está no topo da janela ou na parte inferior, mas não vai mudar se a cabeça mover-se ligeiramente.

É necessário um truque adicional para obter um bom atributo. Como as características de orientação não são afetadas pelo brilho da iluminação, não podemos tratar em especial arestas de alto contraste. Isso significa que as curvas distintas sobre a fronteira de um pedestre são tratadas da mesma forma que detalhes de textura fina na roupa ou no fundo e, assim, o sinal pode ser submerso em ruído. Podemos recuperar a informação do contraste através da contagem de orientações de gradiente com pesos que refletem o quão significativo o gradiente é se comparado a outros gradientes na mesma célula. Vamos escrever $\| \nabla I_x \|$ para a magnitude do gradiente no ponto x da imagem, escrever C para a célula cujo histograma queremos calcular e escrever $w_{x,C}$ para o peso que vamos utilizar para orientação em x nessa célula. Uma escolha natural de peso é

$$w_{\mathbf{x},\mathcal{C}} = \frac{\| \nabla I_{\mathbf{x}} \|}{\sum_{\mathbf{u} \in \mathcal{C}} \| \nabla I_{\mathbf{u}} \|} .$$

Isso compara a magnitude do gradiente aos outros na célula, de modo que os gradientes que são

grandes em comparação com os seus vizinhos têm peso grande. A característica resultante é geralmente chamada de **característica HOG** (de histograma de orientações de gradiente).

Essa construção de característica é a principal forma pela qual a detecção de pedestre difere da detecção de rosto. Caso contrário, construir um detector de pedestre seria muito semelhante à construção de um detector de rosto. O detector varre uma janela através da imagem, calcula recursos para essa janela, em seguida apresenta-a para um classificador. É necessário aplicar a não supressão máxima à saída. Na maior parte das aplicações, a escala e a orientação dos pedestres típicos é conhecida. Por exemplo, conduzindo aplicações em que uma câmera é fixa ao carro, esperamos ver principalmente pedestres na vertical, e estamos interessados apenas em pedestres nas proximidades. Vários conjuntos de dados de pedestres foram publicados e podem ser utilizados para treinar o classificador.

Os pedestres não são o único tipo de objeto que podemos detectar. Na Figura 24.15 vemos que técnicas similares podem ser utilizadas para encontrar uma variedade de objetos em diferentes contextos.



Figura 24.15 Outro exemplo de reconhecimento de objeto usando o recurso SIFT (Scale Invariant Feature Transform), uma versão anterior do recurso HOG. À **esquerda**, imagens de um sapato e um telefone que servem como modelos de objeto. No **centro**, uma imagem de teste. No lado **direito**, o sapato e o telefone foram detectados por: encontrar pontos na imagem, cujas descrições da feição SIFT correspondem ao modelo; cálculo de uma estimativa da pose do modelo; e verificação da estimativa. Normalmente verifica-se uma combinação forte com raros falsos positivos. Imagens de Lowe (1999) © IEEE.

24.4 RECONSTRUÇÃO DO MUNDO EM 3-D

Nesta seção, mostramos como ir da imagem bidimensional para uma representação tridimensional da cena. A questão fundamental é esta: dado que todos os pontos na cena que caem ao longo de um raio para o orifício são projetados para o mesmo ponto na imagem, como recuperar informação tridimensional? Duas ideias vêm em nosso socorro:

- Se tivermos duas (ou mais) imagens a partir de posições de câmera diferentes, podemos triangular para encontrar a posição de um ponto na cena.

- Podemos explorar o histórico do conhecimento sobre a cena física que deu origem à imagem. Dado um modelo de objeto **P**(*Cena*) e uma interpretação do modelo **P**(*Imagem* | *Cena*), podemos calcular uma distribuição posterior **P**(*Cena* | *Imagem*).

Não existe ainda uma teoria unificada para a reconstrução da cena. Fizemos um levantamento de oito pistas visuais utilizadas comumente: **movimento**, **estereopsia binocular**, **múltiplas visões**, **textura**, **sombra**, **contorno** e **objetos familiares**.

24.4.1 Movimento de paralaxe

Se a câmera se move em relação à cena tridimensional, o movimento resultante aparente na imagem, o fluxo óptico, pode ser uma fonte de informação, tanto para o movimento da câmera como para a profundidade na cena. Para entender isso, podemos afirmar (sem provas) uma equação que relaciona o fluxo óptico à velocidade de translação **T** e à profundidade na cena do espectador.

Os componentes do campo de fluxo óptico são

$$v_x(x,y) = \frac{-T_x + xT_z}{Z(x,y)}, \qquad v_y(x,y) = \frac{-T_y + yT_z}{Z(x,y)},$$

onde $Z(x, y)$ é a coordenada z do ponto na cena que corresponde ao ponto na imagem em (x, y).

Observe que ambos os componentes do fluxo óptico, $v_x(x, y)$ e $v_y(x, y)$, são zero no ponto $x = T_x/T_z$, $y = T_y/T_z$. Esse ponto é chamado de **foco de expansão** do campo de fluxo. Suponha que mudemos a origem no plano x-y para ficar no foco de expansão; em seguida, as expressões para o fluxo óptico assumem uma forma particularmente simples. Seja (x', y') as novas coordenadas definidas por $x' = x - T_x/T_z$, $y' = y - T_y/T_z$. Então

$$v_x(x',y') = \frac{x'T_z}{Z(x',y')}, \qquad v_y(x',y') = \frac{y'T_z}{Z(x',y')}.$$

Observe que aqui há uma ambiguidade de fator de escala. Se a câmera estiver se movendo duas vezes mais rápido, cada objeto na cena for duas vezes maior e estiver duas vezes à distância da câmera, o campo de fluxo óptico será exatamente o mesmo. Mas ainda podemos extrair informações bastante úteis.

1. Suponha que você seja uma mosca tentando pousar em uma parede e, à velocidade atual, quer saber o tempo para o pouso. Esse tempo é dado por Z/T_z. Observe que, embora o campo de fluxo óptico instantâneo não possa fornecer a distância Z ou a velocidade do componente T_z, pode fornecer a relação dos dois e, portanto, pode ser usado para controlar a aproximação de pouso. Há evidência experimental considerável de que muitas espécies de animais diferentes exploraram esse palpite.
2. Considere dois pontos às profundidades Z_1, Z_2, respectivamente. Podemos não conhecer o valor

absoluto de qualquer um deles, mas considerando-se o inverso da razão das magnitudes do fluxo óptico nesses pontos podemos determinar a relação de profundidade Z_1/Z_2. Essa é a sugestão de movimento paralaxe, que usamos quando olhamos para fora da janela lateral de um carro em movimento ou trem e infere que as partes mais lentas da paisagem estão mais longe.

24.4.2 Estereopsia binocular

A maioria dos vertebrados têm dois olhos. Isso é útil para a redundância em caso de um olho perdido, mas também ajuda de outras maneiras. A maioria das presas tem olhos no lado da cabeça para permitir um campo mais vasto da visão. Predadores têm os olhos na frente, permitindo-lhes usar a **estereopsia binocular**. A ideia é semelhante ao movimento de paralaxe, exceto que, em vez de usar as imagens ao longo do tempo, usamos duas (ou mais) imagens separadas no espaço. Como determinada característica na cena estará em um lugar diferente em relação ao eixo z de cada plano de imagem, se sobrepusermos as duas imagens, haverá uma **disparidade** no local da característica da imagem nas duas imagens. Você pode ver isso na Figura 24.16, onde o ponto mais próximo da pirâmide é deslocado para a esquerda na imagem da direita e para a direita na imagem da esquerda.

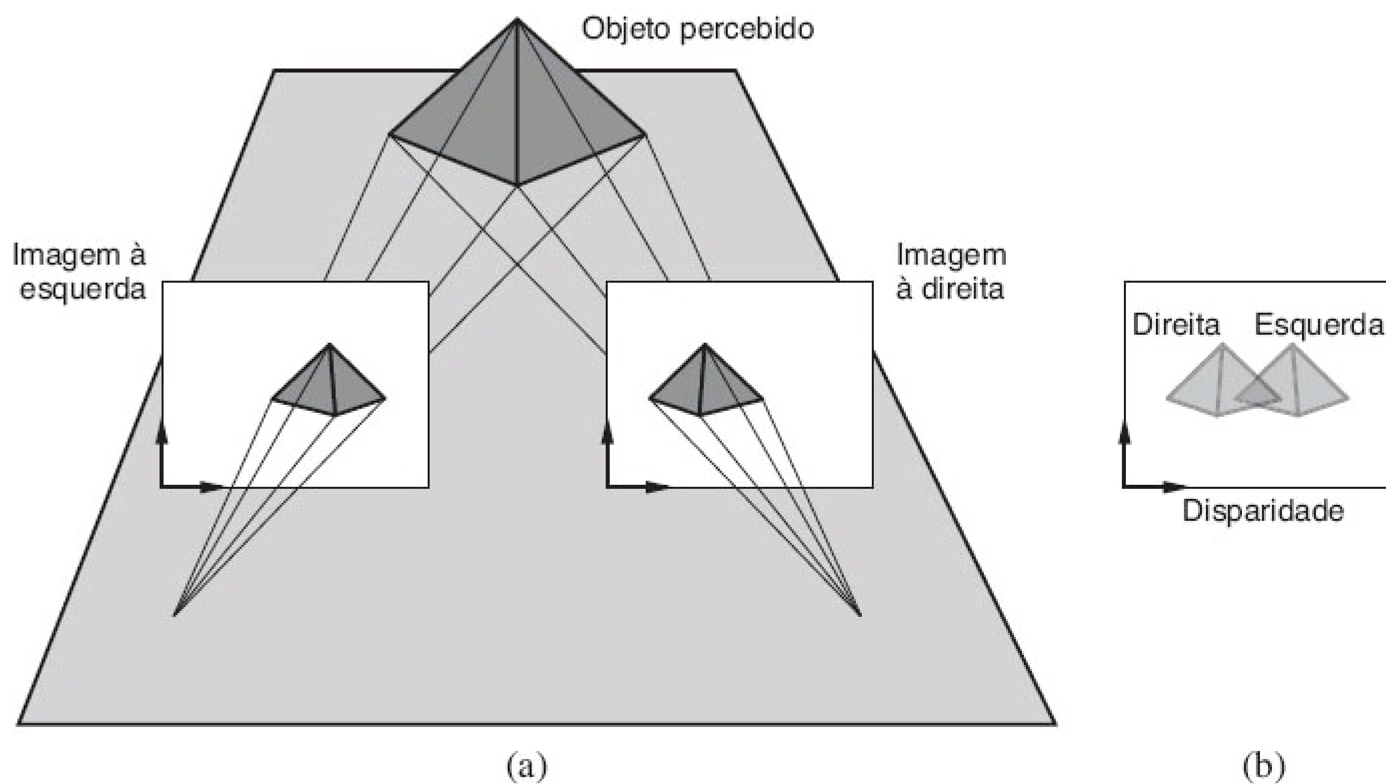


Figura 24.16 A translação de uma câmera paralela em relação ao plano da imagem faz com que recursos de imagem movam-se no plano da câmera. A disparidade de posições resultante é um palpite de profundidade. Se sobrepusermos a imagem esquerda e direita, como em (b), veremos a disparidade.

Observe que, para medir a disparidade, precisamos resolver o problema de correspondência, isto é, determinar um ponto na imagem à esquerda, o ponto na imagem da direita que resulta da projeção do ponto da mesma cena. Isso é análogo ao que se tem de fazer na medição do fluxo óptico, e as abordagens mais simplórias são bastante semelhantes e baseadas na comparação de blocos de pixels

em torno de pontos correspondentes utilizando a soma das diferenças ao quadrado. Na prática, utilizamos algoritmos muito mais sofisticados, que exploram as restrições adicionais.

Partindo do princípio de que podemos medir a disparidade, como isso produz informação sobre a profundidade na cena? Precisaremos trabalhar a relação geométrica entre disparidade e profundidade. Primeiro, vamos considerar o caso em que ambos os olhos (ou câmeras) estão aguardando com interesse com seus eixos ópticos paralelos. A relação da câmera à direita com a câmera à esquerda é, então, apenas um deslocamento ao longo do eixo x por uma quantidade b, a linha de base. Podemos usar as equações de fluxo óptico da seção anterior, se pensarmos nisso como resultado de um vetor de translação **T** agindo no tempo δt com $T_x = b/\delta t$ e $T_y = T_z = 0$. A disparidade horizontal e vertical é dada pelos componentes de fluxo óptico, multiplicada pelo período de tempo δt, $H = v_x \delta t$, $V = v_y \delta t$. Realizando as substituições, obtemos o resultado $H = b/Z$, $V = 0$. Em palavras, a disparidade horizontal é igual à razão entre a linha de base e a profundidade, e a disparidade vertical é zero. Dado que conhecemos b, podemos medir H e recuperar a profundidade Z.

Sob condições normais de visualização, os seres humanos se **fixam**, ou seja, há algum ponto na cena em que os eixos ópticos dos dois olhos se cruzam. A Figura 24.17 mostra dois olhos fixados em um ponto P_0, que está a uma distância Z a partir do meio dos olhos. Por conveniência, vamos calcular a disparidade *angular,* medida em radianos. A disparidade no ponto de fixação P_0 é zero. Para alguns outros pontos P na cena δZ, que é mais distante, podemos calcular os deslocamentos angulares das imagens da esquerda e da direita de P, que chamaremos de P_L e P_R, respectivamente. Se cada uma delas for mostrada por um ângulo de $\delta\theta/2$ em relação a P_0, o deslocamento entre P_L e P_R, que é a disparidade de P, é apenas δq. Da Figura 24.17, $\theta = \frac{b/2}{Z}$ e $\tan(\theta - \delta\theta/2) = \frac{b/2}{Z+\delta Z}$, mas para ângulos pequenos, $\tan\theta \approx \theta$, então

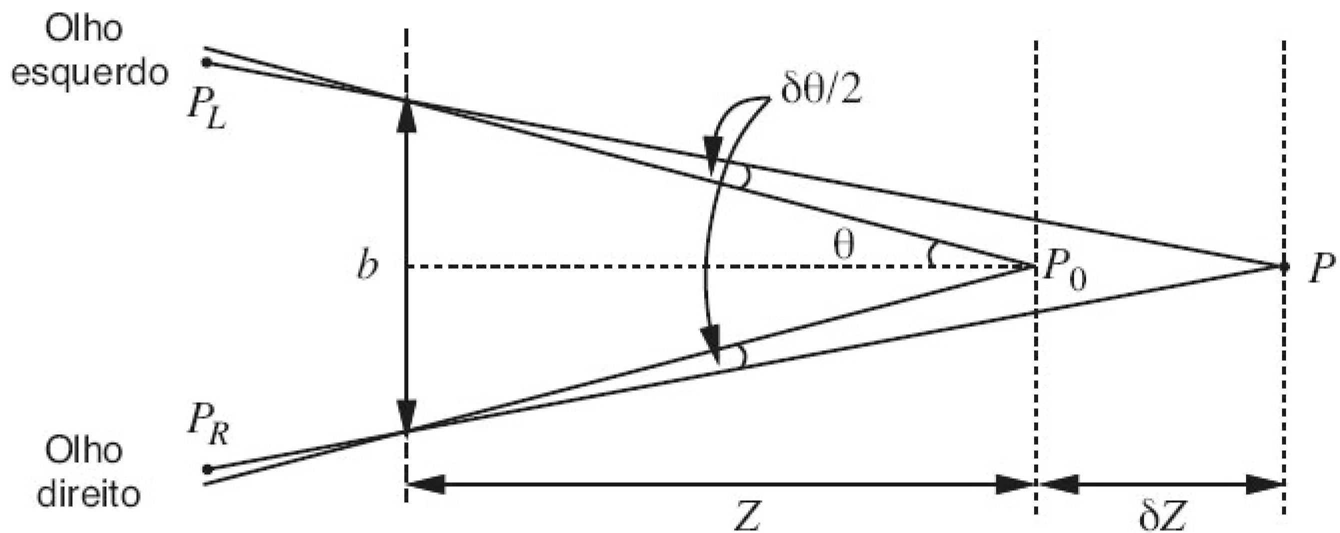


Figura 24.17 Relação entre disparidade e profundidade em estereopsia. Os centros de projeção dos dois olhos são separados de b, e os eixos ópticos cruzam-se no ponto de fixação P_0. O ponto P na cena projeta para os pontos P_L e P_R nos dois olhos. Em termos angulares, a disparidade entre eles é δq. Veja o texto.

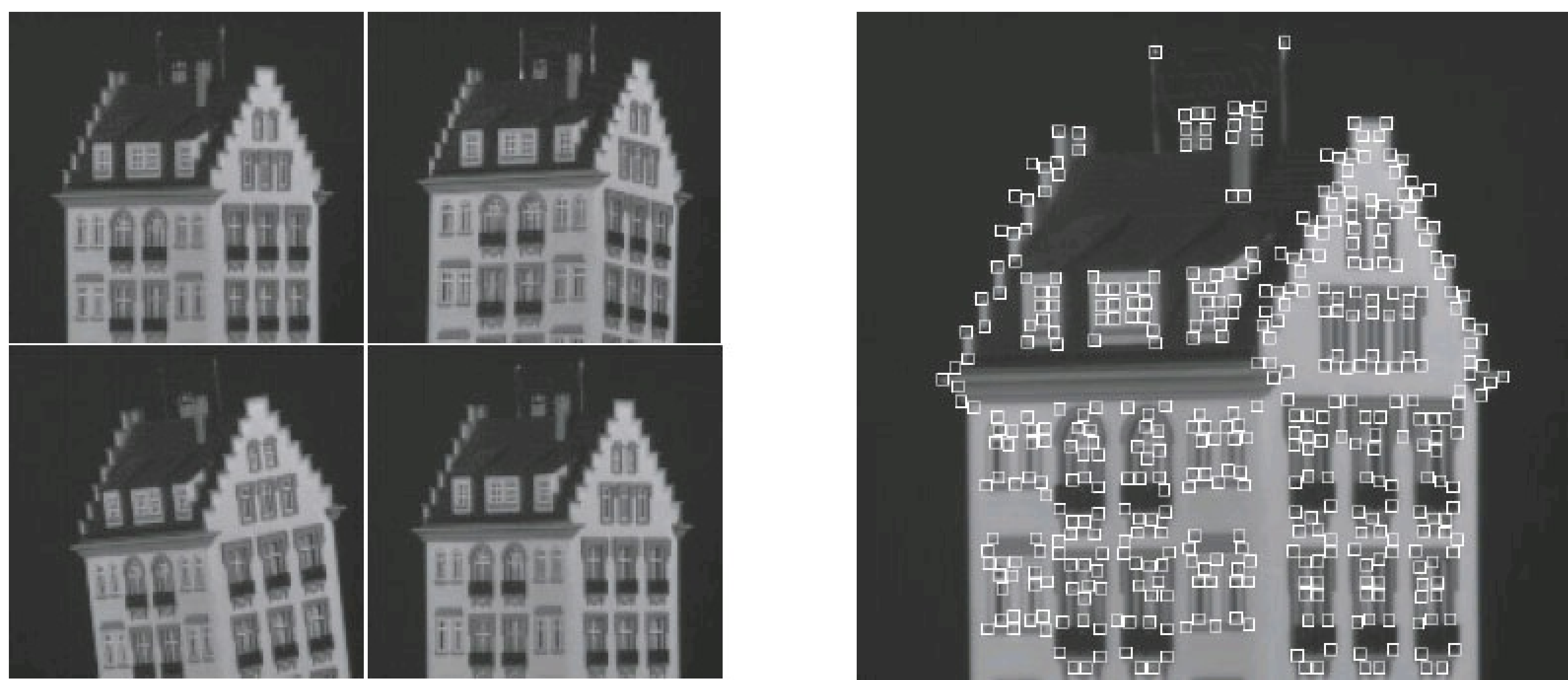
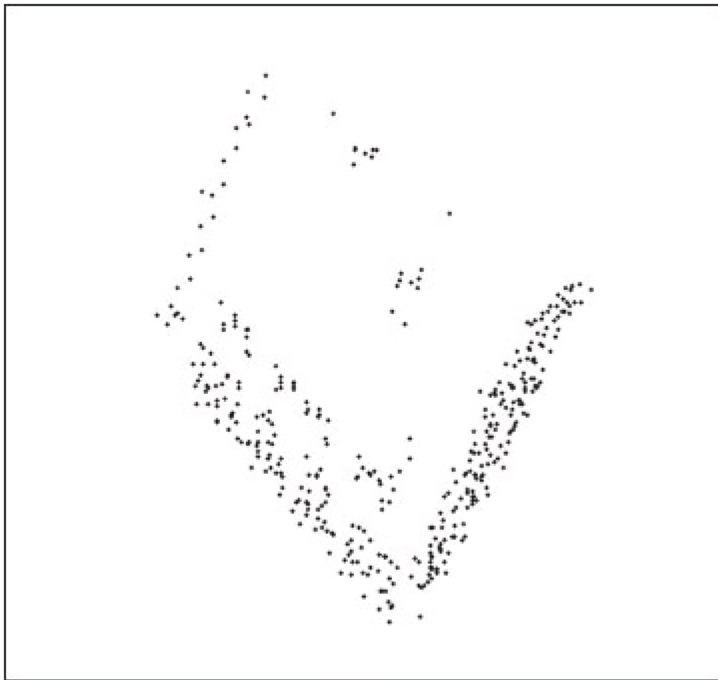


Figura 24.18 (a) Uma sequência de vídeo de quatro quadros em que a câmera se move e gira em relação ao objeto. (b) O primeiro quadro da sequência, com anotações de pequenas caixas destacando as características encontradas pelo detector de recurso. (Cortesia de Carlo Tomasi.)



(a)



(b)

Figura 24.19 (a) Reconstrução tridimensional das localizações dos recursos de imagem na Figura 24.18, mostrada de cima. (b) A casa real, tirada da mesma posição.

$$\delta\theta/2 = \frac{b/2}{Z} - \frac{b/2}{Z + \delta Z} \approx \frac{b\delta Z}{2Z^2}$$

e, uma vez que a disparidade real é δq, temos

$$\text{disparidade} = \frac{b\delta Z}{Z^2}\ .$$

Em seres humanos, b (a distância **base de referência** entre os olhos) é de cerca de 6 cm. Suponha que Z seja cerca de 100 cm. Se o menor δq detectável (correspondente ao tamanho do pixel) for cerca de 5 segundos de arco, isso dá um δZ de 0,4 mm. Para Z = 30 cm, obtemos o valor impressionantemente pequeno δZ = 0,036 mm. Isto é, de uma distância de 30 cm, os seres humanos podem discriminar profundidades que diferem por 0,036 mm, permitindo-nos colocar a linha na agulha e similares.

24.4.3 Múltiplas visualizações

A forma do fluxo óptico ou disparidade binocular são duas instâncias de uma estrutura mais geral, de explorar múltiplas visões para a recuperação da profundidade. Em visão de computador, não há razão em sermos restritos a movimentos diferenciais ou usar apenas duas câmeras convergindo para um ponto de fixação. Assim, foram desenvolvidas técnicas que exploram a informação disponível em múltiplas visões, mesmo a partir de centenas ou milhares de câmeras. Algoritmicamente, existem três subproblemas que precisam ser resolvidos:

- O problema da correspondência, ou seja, identificar características nas imagens diferentes que são projeções da mesma característica no mundo tridimensional.
- O problema da orientação relativa, ou seja, determinar a transformação (rotação e translação) entre os sistemas de coordenadas fixados em câmeras diferentes.
- O problema de estimativa de profundidade, ou seja, a determinação da profundidade de vários pontos no mundo para os quais as projeções do plano de imagem estavam disponíveis em pelo menos duas visões.

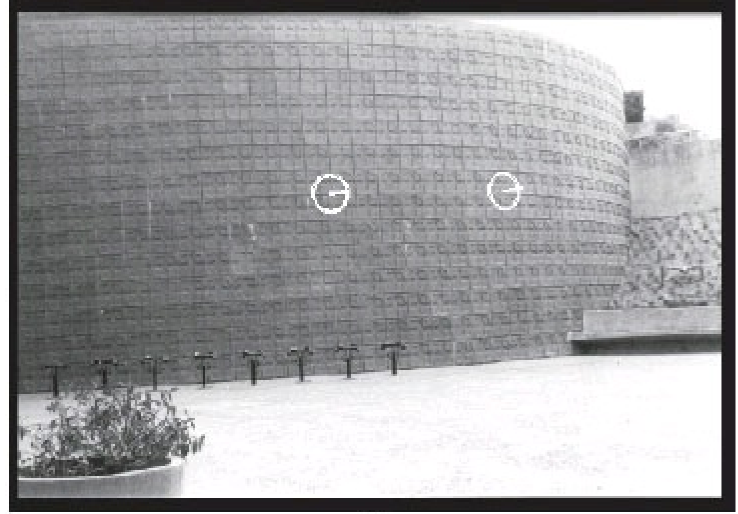
O desenvolvimento de procedimentos sólidos de combinação para o problema de correspondência, acompanhado por algoritmos numericamente estáveis para a solução de orientações relativas e profundidade de cena, é uma das histórias de sucesso de visão computacional. Os resultados dessa abordagem devidos a Tomasi e Kanade (1992) são mostrados nas Figuras 24.18 e 24.19.

24.4.4 Textura

Vimos anteriormente como a textura foi usada para a segmentação de objetos. Também pode ser usada para estimar distâncias. Na Figura 24.20 vemos que uma textura homogênea na cena resulta em elementos diferentes de textura, ou **texels**, na imagem. Todos os mosaicos em (a) são idênticos na cena. Eles parecem diferentes na imagem, por duas razões:



(a)



(b)

Figura 24.20 (a) Cena texturizada. Supõe-se que a textura real seja uniforme e permite a recuperação da orientação da superfície. A orientação da superfície computadorizada é indicada pela sobreposição de um círculo preto e um ponteiro, transformada como se o círculo fosse pintado na superfície naquele ponto. (b) Recuperação da forma de textura de uma superfície curva (círculo branco e ponteiro dessa vez). Imagens de cortesia de Jitendra Malik e Ruth Rosenholtz (1994).

1. *Diferenças nas distâncias dos texels da câmera*. Objetos distantes parecem menores por um fator de escala de $1/Z$.
2. *Diferenças no encurtamento dos texels*. Se todos os texels estiverem no plano de fundo, as distâncias serão vistas em um ângulo que está mais fora da perpendicular e, assim, estarão mais encurtadas. A magnitude do efeito de encurtamento é proporcional ao cos σ, onde σ é a inclinação, o ângulo entre o eixo Z e $\mathbf{n}$, a superfície normal para o texel.

Pesquisadores desenvolveram vários algoritmos que tentam explorar a variação na aparência dos texels projetados como base para determinar superfícies normais. No entanto, a precisão e a aplicabilidade desses algoritmos em nenhum lugar são tão genéricas como as baseadas em utilização de múltiplas visões.

24.4.5 Sombreamento

O sombreamento — a variação na intensidade da luz recebida de diferentes partes de uma superfície em uma cena — é determinado pela geometria da cena e pelas propriedades de refletância das superfícies. Em computação gráfica, o objetivo é calcular o brilho da imagem $I(x, y)$, dada a geometria da cena e as propriedades de refletância dos objetos na cena. A visão do computador visa inverter o processo, isto é, recuperar as propriedades de geometria e de refletância, dado o brilho da imagem $I(x, y)$. É provado que isso é muito difícil de ser feito, a não ser em casos mais simples.

A partir do modelo físico da Seção 24.1.4, sabemos que, se uma superfície normal aponta em direção à fonte de luz, a superfície é brilhante, e se pontos estão longe, a superfície é mais escura. Não podemos concluir que o trecho escuro tem sua normal apontando para fora da luz; em vez disso, poderia ter baixo albedo. Geralmente, o albedo muda muito rapidamente em imagens; o

sombreamento muda muito lentamente, e os seres humanos parecem ser bastante bons em usar essa observação para contar se pouca iluminação, a orientação da superfície ou o albedo fez com que um trecho da superfície ficasse escuro. Para simplificar o problema, vamos supor que o albedo seja conhecido em todos os pontos da superfície. Ainda é difícil recuperar a normal porque o brilho da imagem é uma medição, mas a normal tem dois parâmetros desconhecidos, por isso não podemos simplesmente resolver para a normal. A chave para essa situação parece ser que as normais próximas serão semelhantes, pois a maior parte das superfícies é lisa — não tem mudanças pronunciadas.

A dificuldade real vem de lidar com inter-reflexões. Se considerarmos uma cena interna típica, como os objetos dentro de um escritório, as superfícies não serão iluminadas apenas pela fonte de luz, mas também pela luz refletida de outras superfícies na cena que servem efetivamente como fontes de luz secundária. Esses efeitos de iluminação mútua são bastante significativos e tornam difícil prever a relação entre a normal e o brilho da imagem. Dois trechos de superfície com a mesma normal poderiam ter brilhos muito diferentes porque um recebe a luz refletida de uma grande parede branca e a outra está em frente de apenas uma estante escura. Apesar dessas dificuldades, o problema é importante. Os seres humanos parecem ser capazes de ignorar os efeitos das inter-reflexões e obter uma percepção útil de forma a partir do sombreamento, mas conhecemos pouco sobre os algoritmos que fazem isso.

24.4.6 Contorno

Quando olhamos para um desenho de linha, como na Figura 24.21, temos uma percepção vívida de forma e leiaute tridimensional. Como? É uma combinação de reconhecimento de objetos familiares da cena e a aplicação de restrições genéricas, tais como as seguintes:

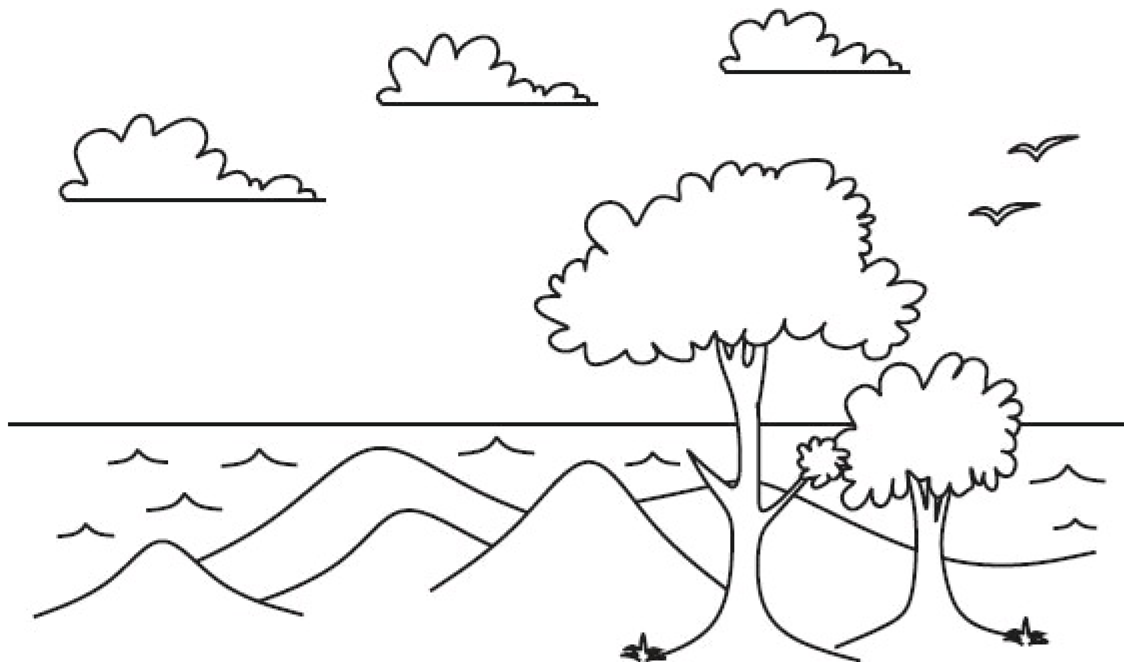


Figura 24.21 Um desenho de linhas evocativas. (Cortesia de Isha Malik.)

- Contornos de oclusão, tais como os contornos das colinas. Um lado do contorno está mais

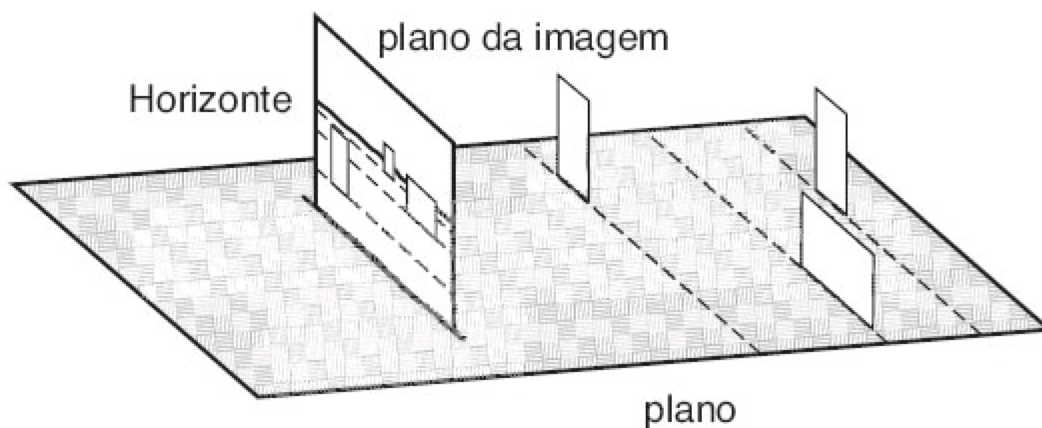
próximo do telespectador, o outro está mais distante. Características tais como convexidade local e simetria fornecem pistas para resolver o problema de **fundo e figura** — atribuição de em que lado do contorno está a figura (mais próximo) e qual é o fundo (mais distante). Em um contorno de oclusão, a linha de visão é tangente à superfície na cena.

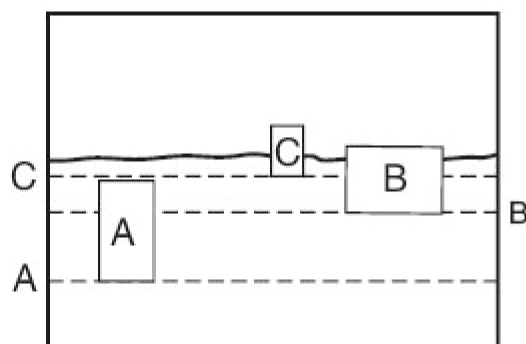
- Junções T. Quando um objeto oclui o outro, o contorno do objeto mais distante é interrompido, assumindo que o objeto mais próximo seja opaco. Uma junção T resulta na imagem.
- Posição no plano de fundo. Os seres humanos, como muitos outros animais terrestres, muitas vezes estão em uma cena que contém um **plano de fundo**, com vários objetos em locais diferentes nesse plano. Devido à gravidade, os objetos típicos não flutuam no ar, mas são suportados por esse plano de fundo e podemos explorar a geometria muito especial desse cenário de visualização.

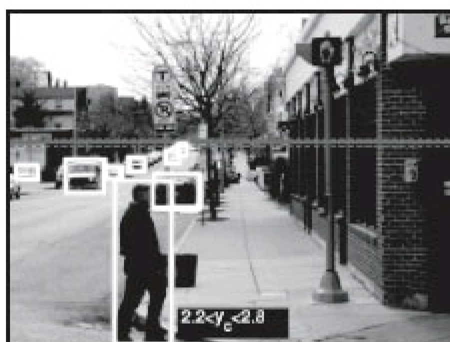
  Vamos trabalhar com a projeção de objetos de alturas diferentes e em locais diferentes no plano de fundo. Suponha que o olho ou a câmera esteja em uma altura h_c acima do plano de fundo. Considere um objeto da altura descansando sobre o plano, cujo fundo está em $(X, -h_c, Z)$ e o topo está em $(X, \delta Y - h_c, Z)$. A parte inferior projeta para o ponto de imagem $(fX/Z, -fh_c/Z)$ e a parte superior para $(fX/Z, f\ (\delta Y - h_c)/Z)$. A parte inferior dos objetos mais próximos (Z pequeno) se projeta para os pontos mais baixos no plano da imagem; os objetos mais distantes têm a parte inferior perto do horizonte.

24.4.7 Objetos e a estrutura geométrica de cenas

A cabeça de um adulto humano típico tem cerca de 20 cm de comprimento. Isso significa que, para alguém a 13 m de distância, o ângulo subtendido para a cabeça na câmera é de 1 grau. Se virmos uma pessoa cuja cabeça parece subtender apenas meio grau, a inferência bayesiana sugere que estamos olhando para uma pessoa normal que está a 26 m de distância, em vez de alguém com a cabeça de metade do tamanho. Essa linha de raciocínio nos fornece um método para verificar os resultados de um detector de pedestres, bem como um método para estimar a distância de um objeto. Por exemplo, todos os pedestres são praticamente da mesma altura e tendem a ficar sobre o plano do chão. Se soubermos onde o horizonte está em uma imagem, podemos classificar os pedestres pela distância até a câmera. Isso funciona porque sabemos onde os pés estão, e os pedestres cujos pés estão mais próximos do horizonte na imagem estão mais afastados da câmera (Figura 24.22). Os pedestres que estão mais distantes da câmera também devem ser menores na imagem. Isso significa que podemos descartar algumas respostas do detector — se um detector encontra um pedestre que é grande na imagem e cujos pés estão perto do horizonte, encontrou um pedestre enorme; isso não existe, de modo que o detector está errado. Na verdade, muitas ou a maioria das janelas de imagem não são janelas de pedestres aceitáveis e não precisam ser apresentadas ao detector.







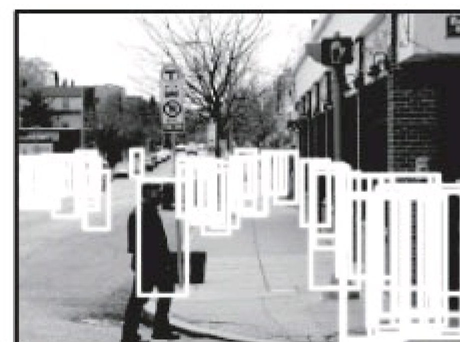


Figura 24.22 Em uma imagem com pessoas em pé sobre um plano no chão, as pessoas cujos pés estão mais próximos do horizonte devem estar mais longe (desenho acima). Isso significa que eles devem parecer menores na imagem (desenho inferior esquerdo). Isso significa que o tamanho e a localização real dos pedestres em uma imagem dependem uns dos outros e da localização do horizonte. Para explorar isso, precisamos identificar o plano do chão, que é feito usando métodos de forma da textura. A partir dessa informação e de alguns pedestres prováveis, podemos recuperar um horizonte como mostrado na imagem central. À direita, caixas de pedestres aceitáveis dão esse contexto geométrico. Observe que os pedestres que na cena são mais altos devem ser mais baixos. Se não forem, são falsos positivos. Imagens de Hoiem *et al.* (2008) © IEEE.

Existem várias estratégias para encontrar o horizonte, incluindo a busca de uma linha de horizonte irregular, com uma porção de azul acima dela, e a utilização da estimativa de orientação da superfície obtida da deformação de textura. Uma estratégia mais elegante explora o reverso de nossas restrições geométricas. Um detector de pedestres que seja razoavelmente confiável é capaz de produzir estimativas do horizonte, se houver muitos pedestres na cena a diferentes distâncias da câmera. Isso ocorre porque a escala relativa dos pedestres é uma pista de onde está o horizonte. Por isso, podemos extrair uma estimativa do horizonte do detector; então usamos essa estimativa para podar os erros do detector de pedestre.

Se o objeto for familiar, pode-se estimar mais do que apenas a distância até ele porque o que ele parece na imagem depende fortemente de sua postura, isto é, sua posição e orientação com respeito ao telespectador. Isso tem muitas aplicações. Por exemplo, em uma tarefa de manipulação industrial, o braço do robô não pode pegar um objeto até que a postura seja conhecida. No caso de objetos rígidos, seja tridimensional ou bidimensional, esse problema tem solução simples e bem definida com base no **método de alinhamento**, que agora desenvolveremos.

O objeto é representado por M características ou pontos distintos m_1, m_2,..., m_M no espaço tridimensional — talvez os vértices de um objeto poliédrico. Eles são medidos em algum sistema de coordenadas que é natural para o objeto. Os pontos são então submetidos a uma rotação **R** tridimensional desconhecida, seguida da translação de uma quantidade desconhecida **t** e projeção para dar origem aos pontos característicos de imagem p_1, p_2,..., p_n no plano da imagem. Em geral, $N \neq M$ porque alguns pontos do modelo podem ser ocluídos, e o detector de atributos poderá perder algumas características (ou inventar características falsas devido ao ruído). Podemos expressar isso como

$$p_i = \Pi(\mathbf{R}m_i + \mathbf{t}) = Q(m_i)$$

para um ponto do modelo m_i e o ponto de imagem correspondente p_1. Aqui, **R** é uma matriz de rotação, **t** é uma translação e Π indica a projeção de perspectiva ou uma de suas aproximações, tal como projeção ortográfica em escala. O resultado líquido é uma transformação Q que trará o ponto de modelo m_i em alinhamento com o ponto de imagem p_i. Embora não conheçamos Q inicialmente, sabemos (para objetos rígidos) que Q deve ser o *mesmo* para todos os pontos do modelo.

Podemos resolver Q, dadas as coordenadas tridimensionais de três pontos do modelo e suas projeções bidimensionais. A intuição é a seguinte: podemos escrever as equações relacionando as coordenadas de p_i às de m_i. Nessas equações, as quantidades desconhecidas correspondem aos parâmetros de rotação da matriz **R** e o vetor de translação **t**. Tendo equações suficientes, devemos ser capazes de resolver Q. Não demonstraremos aqui, apenas afirmaremos o resultado seguinte:

> Dados três pontos não colineares m_1, m_2 e m_3 no modelo e suas projeções ortográficas em escala p_1, p_2 e p_3 no plano de imagem, existem exatamente duas transformações da estrutura coordenada do modelo tridimensional para uma estrutura coordenada de imagem bidimensional.

Essas transformações estão relacionadas por uma reflexão em torno do plano da imagem e podem ser calculadas por uma solução de forma fechada simples. Se pudéssemos identificar as características do modelo correspondentes às três características da imagem, poderíamos calcular Q, a pose do objeto.

Vamos especificar a posição e a orientação em termos matemáticos. A posição de um ponto P na cena é caracterizada por três números, as coordenadas de P (X, Y, Z) em uma estrutura coordenada com a sua origem no orifício e o eixo Z ao longo do eixo óptico (Figura 24.2). O que temos disponível é a projeção em perspectiva (x, y) do ponto na imagem. Isso especifica o raio do orifício ao longo do qual P se encontra; o que não sabemos é a distância. O termo "orientação" poderia ser usado em dois sentidos:

1. **A orientação do objeto como um todo.** Pode-se especificar isso em termos de uma rotação tridimensional relacionando sua estrutura de coordenadas com a câmera.
2. **A orientação da superfície do objeto em P.** Isso pode ser especificado por um vetor normal, **n**, que é um vetor especificando a direção perpendicular à superfície. Muitas vezes, expressamos a orientação da superfície usando as variáveis **obliquidade** e **inclinação**.

Obliquidade é o ângulo entre o eixo Z e **n**. Inclinação é o ângulo entre o eixo X e a projeção de **n** no plano da imagem.

Quando a câmera se move em relação a um objeto, tanto a distância do objeto como a sua orientação mudam. O que é preservado é a **forma** do objeto. Se o objeto for um cubo, esse fato não muda quando o objeto se move. Geômetros tentam formalizar a forma há séculos; o conceito básico é que forma é o que permanece inalterado sob algum grupo de transformações, por exemplo, combinações de rotações e translações. A dificuldade reside em achar uma representação de forma global, que seja geral o suficiente para lidar com a grande variedade de objetos no mundo real — não apenas formas simples, como cilindros, cones e esferas — e ainda podem ser facilmente recuperadas a partir de entrada visual. O problema de caracterizar a forma *local* de uma superfície é muito melhor compreendida. Essencialmente, pode-se fazer isso em termos de curvatura: como é que a superfície normal muda à medida que alguém se move em diferentes direções na superfície? Para um plano, não há nenhuma alteração. Para um cilindro, se alguém se move paralelamente ao eixo, não há mudança, mas na direção perpendicular a superfície normal gira a uma velocidade inversamente proporcional ao raio do cilindro, e assim por diante. Tudo isso é estudado no assunto chamado geometria diferencial.

A forma de um objeto é relevante para algumas tarefas de manipulação (por exemplo, decidir onde agarrar um objeto), mas seu papel mais significativo é o reconhecimento de objetos, em que a forma geométrica, a cor e a textura fornecem as pistas mais importantes para nos permitir identificar objetos, classificar o que está na imagem como exemplo de alguma classe já vista antes, e assim por diante.

24.5 RECONHECIMENTO DE OBJETOS A PARTIR DE INFORMAÇÃO ESTRUTURAL

Colocar uma caixa ao redor de pedestres em uma imagem pode muito bem ser suficiente para evitar o atropelamento. Vimos que podemos encontrar uma caixa reunindo a evidência fornecida por orientações usando métodos de histograma para suprimir o detalhe espacial potencialmente confuso. Se quisermos saber mais sobre o que alguém está fazendo, vamos precisar saber onde estão os seus braços, pernas, corpo e cabeça na imagem. Partes individuais do corpo são muito difíceis de detectar através de um método de janela em movimento porque a sua cor e textura podem variar muito e porque, em geral, são imagens pequenas. Muitas vezes, antebraços e canelas são tão pequenos quanto dois a três pixels de largura. As partes do corpo não costumam aparecer isoladas, e representar o que está ligado ao que poderia ser muito eficaz, pois as partes que são fáceis de encontrar podem nos informar onde procurar as partes que são pequenas e difíceis de detectar.

Inferir o leiaute do corpo humano em imagens é uma tarefa importante em visão porque o leiaute do corpo, muitas vezes, revela o que as pessoas estão fazendo. Um modelo chamado **modelo deformável** pode nos informar quais configurações são aceitáveis: o cotovelo pode dobrar, mas a cabeça nunca se junta ao pé. O modelo deformável mais simples de uma pessoa liga o braço ao antebraço, o braço ao tronco, e assim por diante. Há modelos mais ricos: por exemplo, poderíamos

representar o fato de que os antebraços esquerdo e direito tendem a ter a mesma cor e textura, assim como as pernas esquerda e direita. No entanto, esses modelos mais ricos permanecem difíceis de trabalhar.

24.5.1 A geometria dos corpos: encontrar braços e pernas

Nesse momento, assumimos que sabemos o que parece ser as partes do corpo da pessoa (por exemplo, sabemos a cor e a textura da roupa da pessoa). Podemos modelar a geometria do corpo como uma árvore de onze segmentos (braços e pernas esquerdos e direitos, respectivamente, tronco, rosto e cabelo no topo do rosto), cada um dos quais é retangular. Assumimos que, dada a pose do braço esquerdo, a posição e a orientação (**pose**) do braço inferior esquerdo é independente de todos os outros segmentos; que, dada a postura do torso, a posição do braço esquerdo é independente de todos os segmentos; e estender essas premissas de forma óbvia para incluir o braço direito, as pernas, o rosto e os cabelos. Tais modelos são chamados frequentemente de modelos de “pessoas de papelão”. Os modelos formam uma árvore, que normalmente é enraizada no torso. Vamos procurar a imagem que tenha a melhor correspondência para essa pessoa de papelão usando métodos de inferência para uma rede de Bayes de estruturada em árvore (veja o Capítulo 14).

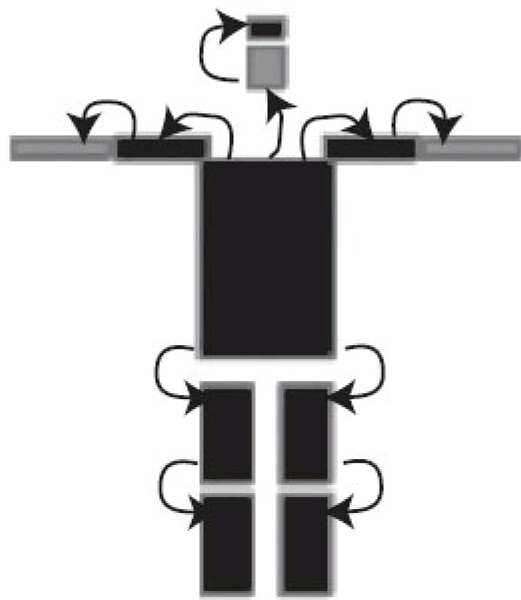
Existem dois critérios para avaliar uma configuração. Primeiro, um retângulo de imagem deve parecer com o seu segmento. Por enquanto, ficará vago o que isso significa exatamente, mas assumiremos que temos uma função ϕ_i que pontua o quão bem um retângulo de imagem corresponde a um segmento do corpo. Para cada par de segmentos relacionados, temos outra função ψ que pontua o quão bem a relação entre um par de retângulos de uma imagem combina com as que são esperadas dos segmentos do corpo. As dependências entre os segmentos formam uma árvore, de modo que cada segmento tem apenas um pai, e poderíamos escrever $\psi_{i,\mathbf{pa}(i)}$. Todas as funções serão maiores se a correspondência for melhor; assim, podemos considerá-las como sendo uma probabilidade de log. Então, o custo de uma combinação particular que aloca a imagem retângulo m_i para o segmento do corpo i é

$$\sum_{i \in \text{segmentos}} \phi_i(m_i) + \sum_{i \in \text{segmentos}} \psi_{i,\text{pa}(i)}(m_i, m_{\text{pa}(i)}) \,.$$

A programação dinâmica pode encontrar a melhor correspondência porque o modelo relacional é uma árvore.

É inconveniente buscar em um espaço contínuo, e vamos discretizar o espaço da imagem dos retângulos. Discretizamos a localização e a orientação dos retângulos de tamanho fixo (os tamanhos podem ser diferentes para segmentos diferentes). Como os tornozelos e os joelhos são diferentes, precisamos distinguir entre um retângulo e o mesmo retângulo com um giro de 180º. Pode-se visualizar o resultado como um conjunto de pilhas muito grandes de pequenos retângulos de imagem, com corte em locais e orientações diferentes. Há uma pilha por segmento. Devemos agora encontrar a melhor alocação de retângulos aos segmentos. Isso é lento porque há muitas imagens de retângulos e, para o modelo que demos, escolher o tronco correto será $O(M^6)$, se houver M imagens de retângulos.

No entanto, várias acelerações estão disponíveis para uma escolha apropriada de ψ, e o método é prático (Figura 24.23). O modelo normalmente é conhecido como **modelo de estrutura pictórico**.



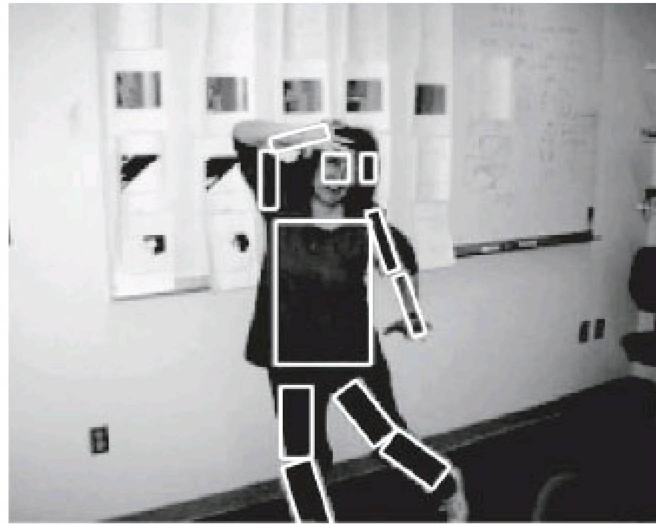
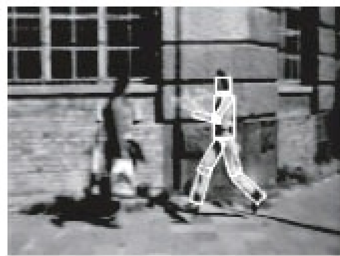
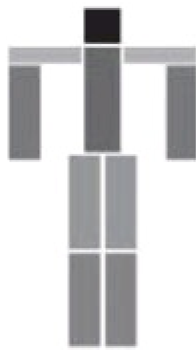
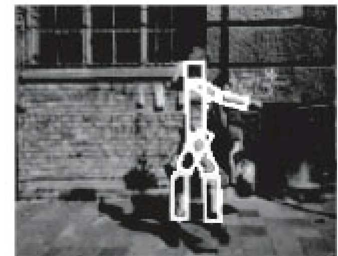


Figura 24.23 Um modelo de estrutura pictórica avalia uma combinação entre um conjunto de retângulos de imagem e uma pessoa de papelão (mostrado à esquerda), marcando a semelhança na aparência entre os segmentos do corpo e os segmentos de imagem e as relações espaciais entre os segmentos de imagem. Geralmente, uma combinação será melhor se os segmentos de imagem tiverem a aparência correta e estiverem no lugar certo com relação um ao outro. O modelo de aparência utiliza cores médias para cabelo, cabeça, tronco, braços e pernas. As relações relevantes são mostradas com setas. À direita, a melhor correspondência para determinada imagem obtida ao utilizar programação dinâmica. A combinação é uma estimativa razoável da configuração do corpo. Figura de Felzenszwalb e Huttenlocher (2000) © IEEE.









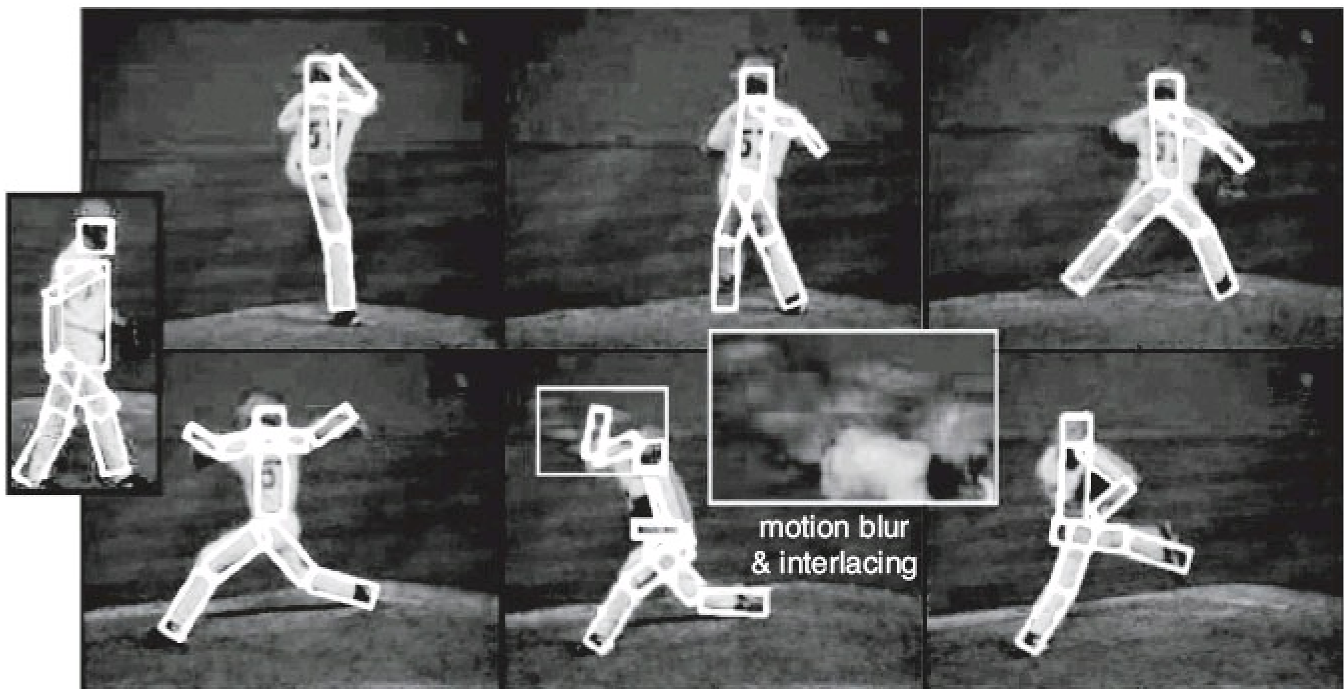


Figura 24.24 Podemos rastrear o deslocamento de pessoas com um modelo de estrutura pictórica, primeiro obtendo um modelo de aparência, depois o aplicando. Para obter o modelo de aparência, digitalizamos a imagem para encontrar uma postura de caminhada lateral. O detector não precisa ser muito preciso, mas deve produzir poucos falsos positivos. A partir da resposta do detector, podemos ler os pixels que se encontram em cada segmento do corpo e outros que não se encontram sobre esse segmento. Isso torna possível construir um modelo discriminativo do aparecimento de cada parte do corpo, e eles são amarrados em um modelo de estrutura pictórica da pessoa que está sendo rastreada. Finalmente, podemos traçar de maneira segura detectando esse modelo em cada quadro. À medida que os quadros na parte inferior da imagem sugerem, esse procedimento pode rastrear configurações de corpo complicadas, de mudanças rápidas, apesar da degradação do sinal de vídeo devido ao *motion blur*. Figura de Ramanan *et al*. (2007) © IEEE.

Lembre-se da nossa suposição de que sabemos o que precisamos saber sobre como a pessoa parece. Se estivermos combinando uma pessoa em uma única imagem, o recurso mais útil para pontuar combinações do segmento acaba por ser a cor. Características de textura não funcionam bem, na maioria dos casos, porque as dobras em roupas folgadas produzem padrões de sombreamento

forte que sobrepõem a textura da imagem. Esses padrões são fortes o suficiente para interromper a textura verdadeira da roupa. Em trabalho recentes, ψ reflete tipicamente a necessidade de as extremidades dos segmentos estarem razoavelmente próximas, mas normalmente não há restrições sobre os ângulos. Geralmente, não sabemos com o que uma pessoa se parece, e deve-se construir um modelo de aparências do segmento. Chamamos a descrição do que uma pessoa se parece como **modelo de aparência**. Se tivermos de relatar a configuração de uma pessoa em uma única imagem, poderemos iniciar com um modelo de aparência mal sintonizado, estimar a configuração com ele e então reestimar a aparência, e assim por diante. No vídeo, temos muitos quadros da mesma pessoa, e isso vai revelar a sua aparência.

24.5.2 Aparência coerente: rastreamento de pessoas em vídeo

O rastreamento de pessoas no vídeo é um problema prático importante. Se pudéssemos relatar de forma confiável a localização dos braços, pernas, tronco e cabeça em sequências de vídeo, poderíamos construir muitas interfaces de jogo melhoradas e sistemas de vigilância. Os métodos de filtragem não tiveram muito sucesso com esse problema porque as pessoas podem produzir acelerações grandes e se movem muito rápido. Isso significa que, para um vídeo de 30 Hz, a configuração do corpo no quadro i não restringe a configuração do corpo no quadro i +1. Atualmente, os métodos mais eficazes exploram o fato de que a aparência muda muito lentamente a partir do quadro a quadro. Se pudéssemos inferir um modelo de aparência de um indivíduo de um vídeo, poderíamos usar essa informação em um modelo de estrutura pictórica para detectar essa pessoa em cada quadro do vídeo. Poderíamos então ligar essas localizações ao longo do tempo para rastrear.

Existem várias maneiras de inferir um bom modelo de aparências. Consideramos o vídeo como uma grande pilha de fotos da pessoa que desejamos rastrear. Podemos explorar essa pilha procurando por modelos de aparências que explicam muitas das imagens. Isso funcionaria através da detecção de segmentos do corpo em cada quadro usando o fato de que os segmentos têm arestas mais ou menos paralelas. Tais detectores não são particularmente confiáveis, mas os segmentos que queremos encontrar são especiais. Eles vão aparecer pelo menos uma vez na maioria dos quadros de vídeo; tais segmentos podem ser encontrados agrupando-se as respostas do detector. É melhor começar com o tronco porque é grande e porque os detectores de tronco tendem a ser confiáveis. Uma vez que temos um modelo de aparência do tronco, os segmentos superiores de perna, e assim por diante, deverão aparecer perto do tronco. Esse raciocínio gera um modelo de aparência, mas pode não ser confiável se as pessoas aparecerem contra um fundo quase fixo, onde o detector de segmento gera muitos falsos positivos. Uma alternativa é estimar a aparência para muitos dos quadros de vídeo, reestimando diversas vezes a configuração e a aparência; vemos então se um modelo de aparência explica muitos quadros. Uma alternativa bastante confiável na prática é a aplicação de um detector para uma configuração de corpo fixo para todos os quadros. Uma boa opção de configuração é a que é fácil de detectar de forma confiável, e onde há uma chance forte de a pessoa aparecer nessa configuração mesmo em uma sequência curta (caminhadas laterais é uma boa escolha). Ajustamos o detector para ter uma taxa de falso positivo baixa; assim sabemos quando ele responde que encontramos uma pessoa real; e, como localizamos o seu tronco, braços, pernas e cabeça, sabemos como esses segmentos se parecem.

Se os sistemas de visão pudessem analisar o vídeo e compreender o que as pessoas estão fazendo, seríamos capazes de projetar melhor edifícios e lugares públicos, recolher e utilizar dados sobre o que as pessoas fazem em público; construir sistemas de vigilância mais precisos, mais seguros e menos intrusivos; construir comentaristas esportivos computadorizados; e construir interfaces homem-computador que observassem as pessoas e reagissem ao seu comportamento. Aplicações de interfaces reativas vão de jogos de computador que fazem um personagem levantar e mover-se até sistemas que economizam energia através do gerenciamento de calor e luz em um prédio ao encontrar onde estão os ocupantes e o que estão fazendo.

Alguns problemas são bem compreendidos. Se as pessoas são relativamente pequenas no quadro de vídeo, e o fundo é estável, é fácil detectar as pessoas, subtraindo uma imagem de fundo a partir do quadro atual. Se o valor absoluto da diferença é grande, essa **subtração do fundo** declara o pixel como sendo um pixel de imagem de frente; ao ligar *blobs* de imagens de frente ao longo do tempo, obtém-se uma pista.

Alguns comportamentos estruturados como balé, ginástica ou *tai chi* têm vocabulários específicos de ações. Quando realizados contra um fundo simples, os vídeos dessas ações são fáceis de lidar. A subtração de fundo identifica as principais regiões em movimento, e podemos construir características HOG (mantendo o controle do fluxo, em vez da orientação) para apresentar para um classificador. Podemos detectar padrões consistentes de ação com uma variante do nosso detector de pedestres, onde as características de orientação são recolhidas em histogramas ao longo do tempo e do espaço (Figura 24.25).



Figura 24.25 Algumas ações humanas complexas produzem padrões consistentes de aparência e movimento. Por exemplo, beber envolve movimentos da mão em frente do rosto. As três primeiras imagens são detecções corretas de beber, a quarta é um falso positivo (o cozinheiro está olhando para o pote de café, mas não bebendo). Figura de Laptev e Perez (2007) © IEEE.

Problemas mais gerais permanecem em aberto. A grande questão de pesquisa é associar as observações do corpo, os objetos próximos aos objetivos e as intenções das pessoas em movimento. Uma fonte de dificuldade é a falta de um vocabulário simples do comportamento humano. O comportamento é como a cor, em que as pessoas tendem a pensar que conhecem uma porção de nomes de comportamento, mas não conseguem produzir longas listas de tais palavras quando necessário. Há bastante evidência de que os comportamentos combinam — você pode, por exemplo, beber um *milk-shake* dentro de um caixa eletrônico —, mas ainda não sabemos quais são as peças, como funciona a composição ou quantas combinações pode haver. Uma segunda fonte de dificuldade é que não sabemos que características expõe o que está acontecendo. Por exemplo, saber que alguém

está perto de um caixa eletrônico pode ser o suficiente para dizer que ele está entrando no caixa eletrônico. A terceira dificuldade é que o raciocínio habitual sobre a relação entre dados de treinamento e de teste não é confiável. Por exemplo, não podemos argumentar que um detector de pedestres é seguro simplesmente porque tem bom desempenho em um grande conjunto de dados porque esse conjunto de dados pode muito bem omitir fenômenos importantes, mas raros (por exemplo, pessoas utilizando bicicletas). Não gostaríamos que nosso motorista automatizado atropelasse um pedestre que passou a fazer algo incomum.

24.6.1 Palavras e imagens

Muitos sites oferecem coleções de imagens para visualização. Como podemos encontrar as imagens que queremos? Vamos supor que o usuário insira uma consulta de texto como “corrida de bicicleta”. Algumas das imagens terão palavras-chave ou legendas em anexo, ou virão de páginas da Web que contêm texto próximo à imagem. Para esses, a recuperação da imagem pode ser como a recuperação de texto: ignora as imagens e combina o texto da imagem em relação à consulta (veja a Seção 22.3).

No entanto, as palavras-chave geralmente são incompletas. Por exemplo, a imagem de um gato brincando na rua pode ser marcada com palavras como “gato” e “rua”, mas é fácil esquecer-se de mencionar o “lixo” ou as “espinhas de peixe”. Assim, uma tarefa interessante é anotar uma imagem (que pode ter algumas palavras-chave) com mais palavras-chave adequadas.

Na versão mais simples dessa tarefa, temos um conjunto de imagens de exemplo corretamente marcadas e queremos marcar algumas imagens de teste. Às vezes, esse problema é conhecido como autoanotação. As soluções mais precisas são obtidas utilizando o método dos vizinhos mais próximos. Encontram-se as imagens de treinamento que estão mais próximas da imagem de teste em um espaço de características métricas que é treinado por exemplos e, então, relatam-se suas identificações.

Outra versão do problema envolve predizer que marcações anexar a quais regiões em uma imagem de teste. Aqui não sabemos que regiões produziram quais identificações para os dados de treinamento. Podemos usar uma versão de maximização da expectativa para adivinhar uma correspondência inicial entre o texto e as regiões, e então estimar uma decomposição melhor em regiões, e assim por diante.

24.6.2 Reconstrução a partir de muitas visões

A estereopsia binocular funciona porque, para cada ponto, temos quatro medidas restringindo três graus de liberdade desconhecidos. As quatro medidas são as posições (x, y) do ponto em cada visão, e os graus desconhecidos de liberdade são os valores das coordenadas (x, y, z) do ponto na cena. Esse argumento geral sugere, corretamente, que existem restrições geométricas que impedem a maioria dos pares de pontos de serem correspondências aceitáveis. Muitas imagens de um conjunto de pontos deveriam revelar as suas posições de forma inequívoca.

Nem sempre é preciso uma segunda imagem para obter uma segunda visão de um conjunto de pontos. Se acreditarmos que o conjunto de pontos original vem de um objeto familiar 3-D rígido, poderemos ter um modelo de objeto disponível como fonte de informação. Se esse modelo de objeto consiste em um conjunto de pontos 3-D ou um conjunto de imagens do objeto, e se pudermos estabelecer as correspondências dos pontos, poderemos determinar os parâmetros da câmera que produziram os pontos na imagem original. Essa é uma informação muito poderosa. Poderíamos usá-la para avaliar a nossa hipótese inicial de que os pontos vêm de um modelo de objeto. Fazemos isso usando alguns pontos para determinar os parâmetros da câmera, em seguida projetando pontos de modelo nessa câmera e verificando para constatar se há pontos de imagem nas proximidades.

Esboçamos aqui uma tecnologia que agora está muito bem desenvolvida. A tecnologia pode ser generalizada para lidar com visões que não são ortográficas; para lidar com pontos que são observados em apenas algumas visões; para lidar com propriedades da câmera desconhecidas, como o comprimento focal; para explorar várias buscas sofisticadas para correspondências apropriadas; e para fazer a reconstrução a partir de grande número de pontos e de visões. Se a localização dos pontos nas imagens for conhecida com alguma precisão, e as direções de visão forem razoáveis, pode-se obter uma câmera de precisão muito alta e informações do ponto. Algumas aplicações são

- **Construção de modelo**: por exemplo, pode-se construir um sistema de modelagem que toma uma sequência de vídeo e produz uma estrutura entrelaçada de polígonos texturizados tridimensional, muito detalhada, para utilização em aplicações de computação gráfica e de realidade virtual. Modelos como esse podem agora ser construídos a partir de conjuntos de imagens aparentemente nada promissores. Por exemplo, a Figura 24.26 mostra um modelo da Estátua da Liberdade construída a partir de imagens encontradas na Internet.

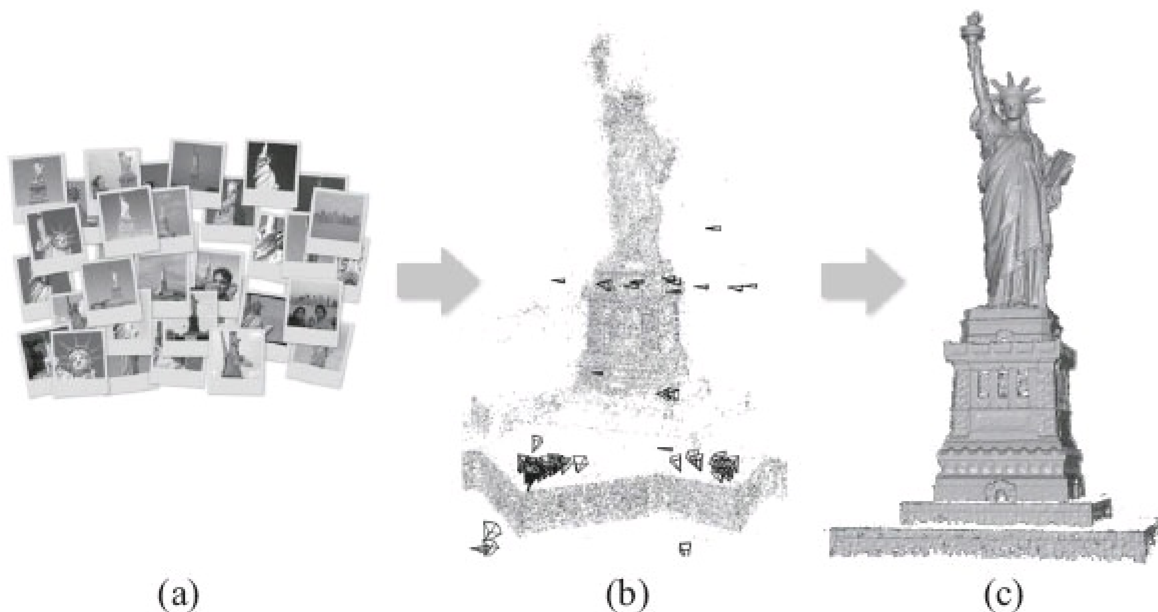


Figura 24.26 A última geração em reconstrução de várias vistas está hoje muito avançada. Essa figura descreve um sistema construído por Michael Goesele e colegas da Universidade de Washington, TU Darmstadt, e Microsoft Research. De uma coleção de imagens de um monumento tomada por uma grande comunidade de usuários e postadas na Internet (a), seu sistema pode determinar as direções de visualização dessas imagens, mostradas por pirâmides pretas pequenas em (b) e por uma reconstrução abrangente 3-D em (c).

- **Casamento de movimentos**: para colocar personagens de computação gráfica em vídeo real, precisamos saber como a câmera se moveu para o vídeo real, para que possamos representar o personagem corretamente.
- **Reconstrução de caminho**: robôs móveis precisam saber onde estavam. Se estiverem se movendo em um mundo de objetos rígidos, executar a reconstrução e manter a informação da câmera é uma maneira de obter o caminho.

24.6.3 Utilizar a visão para controlar o movimento

Uma das principais aplicações da visão é de fornecer informações tanto para a manipulação de objetos — pegá-los, segurá-los, girá-los, e assim por diante — como para a navegação enquanto se evita obstáculos. A capacidade de usar a visão para esses fins está presente nos sistemas visuais mais primitivos do animal. Em muitos casos, o sistema visual é mínimo, no sentido de que extrai do campo de luz disponível apenas a informação que o animal precisa para informar o seu comportamento. Muito provavelmente, os sistemas de visão moderna evoluíram desde o início, organismos primitivos que usavam um ponto fotossensível em uma das extremidades para se orientar na direção da luz (ou para longe dela). Vimos, na Seção 24.4, que as moscas utilizam um sistema de detecção de fluxo ótimo muito simples para aterrissar em paredes. Um estudo clássico, *What the Frog's Eye Tells the Frog's Brain* (Lettvin *et al*., 1959), observa sobre um sapo que "Ele vai morrer de fome cercado por alimentos se eles não estiverem se movendo. Sua escolha de alimento é determinada apenas pelo tamanho e pelo movimento".

Vamos considerar um sistema de visão para um veículo automatizado dirigindo em uma autoestrada. As tarefas enfrentadas pelo motorista incluem o seguinte:

1. Controle lateral — assegura que o veículo permaneça seguro dentro de sua pista ou troca de pista suavemente quando necessário.
2. Controle longitudinal — garante que haja uma distância segura do veículo da frente.
3. Esquiva de obstáculos — monitora veículos nas pistas vizinhas e está preparado para manobras evasivas, se um deles decidir mudar de faixa.

O problema para o motorista é gerar direção adequada, aceleração e ações de frenagem para melhor realizar essas tarefas.

Para o controle lateral, é preciso manter uma representação da posição e orientação do carro em relação à pista. Podemos usar algoritmos de detecção de arestas para encontrar as arestas correspondentes aos segmentos marcadores de pista. Podemos, então, ajustar as curvas suaves a esses elementos de aresta. Os parâmetros dessas curvas levam informações sobre a posição lateral do carro, a direção que ele aponta em relação à pista e a curvatura da pista. Essa informação, juntamente com informações sobre a dinâmica do carro, é tudo o que é necessário para o sistema de controle de direção. Se tivermos bons mapas detalhados da estrada, o sistema de visão servirá para confirmar nossa posição (e observar os obstáculos que não estão no mapa).

Para controle longitudinal, é preciso conhecer as distâncias dos veículos da frente. Isso pode ser

realizado com estereopsia binocular ou fluxo óptico. Usando essas técnicas, carros controlados por visão podem agora dirigir de forma confiável em estrada de velocidades.

O caso mais geral de robôs móveis navegando em vários ambientes interiores e exteriores também tem sido estudado. Um problema particular, a localização do robô em seu ambiente, agora tem soluções muito boas. Um grupo de Sarnoff desenvolveu um sistema baseado em duas câmeras procurando pontos com características de rastro em 3-D e usa isso para reconstruir a posição do robô em relação ao meio ambiente. Na verdade, existem dois sistemas de câmeras estereoscópicas, uma que olha para a frente e outra que olha para trás — isso dá maior robustez, caso o robô tenha de passar por um trecho sem características devido às sombras escuras, paredes brancas, e assim por diante. É improvável que não haja características na frente ou atrás. Agora, é claro que isso pode acontecer, então é providenciada uma cópia de segurança usando uma unidade de movimento inercial (UMI) um pouco parecida com os mecanismos para a detecção de aceleração que nós humanos temos em nosso ouvido interno. Através da integração da aceleração sentida duas vezes, pode-se acompanhar a mudança de posição. Combinar os dados da visão com os da UMI é um problema de fusão de evidência probabilística e pode ser atacado utilizando técnicas como filtragem de Kalman, que estudamos em outras partes do livro.

No uso de odometria visual (estimativa da mudança de posição), como em outros problemas de odometria, há o problema de "integração de informação incremental do movimento", que envolve erros posicionais acumulando-se ao longo do tempo. A solução para isso é usar pontos de referência para fornecer correções da posição absoluta: assim que o robô passar por um local em seu mapa interno, pode ajustar a estimativa de sua posição adequadamente. Com essa técnica foram demonstradas precisões da ordem de centímetros.

O exemplo da condução torna um ponto muito claro: *para uma tarefa específica, não é preciso recuperar toda a informação que, em princípio, pode ser recuperada de uma imagem.* Não é necessário recuperar a forma exata de cada veículo, resolver por forma da textura da superfície da grama adjacente à autoestrada, e assim por diante. Em vez disso, um sistema de visão deve calcular exatamente o que é necessário para realizar a tarefa.

24.7 RESUMO

Embora a percepção pareça ser uma atividade sem esforço para seres humanos, ela exige quantidade significativa de computação sofisticada. A meta da visão é extrair informações necessárias para tarefas como manipulação, navegação e reconhecimento de objetos.

- O processo de **formação de imagens** é bem compreendido em seus aspectos geométricos e físicos. Dada uma descrição de uma cena tridimensional, podemos produzir facilmente um quadro dessa cena a partir de alguma posição arbitrária da câmera (o problema dos gráficos). Inverter o processo, indo de uma imagem até uma descrição da cena, é mais difícil.
- Para extrair as informações visuais necessárias para as tarefas de manipulação, navegação e reconhecimento, devem ser construídas representações intermediárias. Os primeiros algoritmos de **processamento de imagens** extraem características primitivas da imagem, como arestas e

regiões.

- Existem várias sugestões na imagem que permitem obter informações tridimensionais sobre a cena: análise de movimento, visão estereoscópica, textura, sombreamento e contorno. Cada uma dessas sugestões conta com suposições práticas sobre cenas físicas, a fim de fornecer interpretações quase não ambíguas.
- O reconhecimento de objetos em sua generalidade completa é um problema muito difícil. Descrevemos abordagens baseadas no brilho e nas características. Também apresentamos um algoritmo simples para avaliação de poses. Existem outras possibilidades.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O olho foi desenvolvido na explosão do Cambriano (530 milhões de anos atrás), aparentemente em um ancestral comum. Desde então, infinitas variações se desenvolveram em criaturas diferentes, mas o mesmo gene, Pax-6, regula o desenvolvimento do olho em animais tão diversos como seres humanos, camundongos e *Drosophila*.

As tentativas sistemáticas de compreender a visão humana remontam a tempos antigos. Euclides (cerca de 300 a.C.) escreveu sobre a perspectiva natural — o mapeamento que associa a cada ponto P no mundo tridimensional a direção do raio OP que une o centro da projeção O ao ponto P. Ele também estava consciente da noção de paralaxe de movimento. O uso da perspectiva na arte foi desenvolvido na cultura romana antiga, como evidenciado pela arte encontrada nas ruínas de Pompeia (79 d.C.), mas então ficou perdida por cerca de 1.300 anos.

A compreensão matemática da projeção em perspectiva, dessa vez no contexto da projeção sobre superfícies planas, teve outro avanço significativo no século XV, na Itália, à época do Renascimento.

A Brunelleschi (1413) normalmente é creditada a criação das primeiras pinturas baseadas em projeção geometricamente correta de uma cena tridimensional. Em 1435, Alberti codificou as regras e inspirou gerações de artistas cujas realizações nos deixam maravilhados até hoje. Particularmente notáveis em seu desenvolvimento da ciência da perspectiva, como era conhecida na época, foram Leonardo da Vinci e Albrecht Dürer. As descrições do final do século XV feitas por Leonardo sobre a interação entre luz e sombra (*chiaroscuro*), regiões de sombra e penumbra e a perspectiva aérea ainda merecem ser lidas em sua tradução (Kemp, 1989). Stork (2004) analisou a criação de várias peças de arte da Renascença utilizando técnicas de visão de computador.

Embora a perspectiva fosse conhecida pelos gregos, eles estavam curiosamente confusos com o papel dos olhos na visão. Aristóteles imaginou os olhos como dispositivos que emitiam raios, à maneira dos localizadores a *laser* modernos. Essa visão equivocada foi derrubada pelos trabalhos dos cientistas árabes, como Abu Ali Alhazen no século X. Alhazen também desenvolveu a *câmera obscura*, uma sala (*câmera* é uma palavra latina para "sala" ou "quarto") com um orifício que projeta uma imagem na parede oposta. É claro que, em todas essas câmeras, a imagem estava invertida, o que causava grande confusão. Se o olho fosse considerado um dispositivo de formação de imagem desse tipo, como explicar o fato de não vermos imagens invertidas? Esse enigma exercitou as maiores mentes da época (inclusive Leonardo). Kepler propôs a princípio que a lente do olho focaliza uma imagem na retina, e Descartes cirurgicamente removeu um olho de boi e

demonstrou que Kepler estava certo. Havia ainda perplexidade quanto à razão de não vermos tudo de cabeça para baixo; hoje sabemos que isso é apenas uma questão de acesso à estrutura de dados da retina da maneira correta.

Na primeira metade do século XX, os resultados de pesquisa mais significativos na visão foram obtidos pela escola de psicologia Gestalt, liderada por Max Wertheimer. Apontou-se a importância da organização da percepção: para um observador humano, a imagem não é uma coleção de saídas fotorreceptoras pontilistas (pixels em terminologia de visão computacional), mas é organizada em grupos coerentes. Pode-se traçar a motivação em visão por computador de retornar as regiões e curvas a esse discernimento. Os gestaltistas também chamaram a atenção para o fenômeno da "figura de fundo" — um contorno que separa duas regiões de imagem que, no mundo, estão em profundidades diferentes, parecem pertencer apenas à região mais próxima, a "figura", e não à região mais longínqua, o "fundo". O problema de visão por computador de classificar curvas de imagem de acordo com sua importância no cenário pode ser imaginado como uma generalização desse discernimento.

O período após a Segunda Guerra Mundial foi marcado por uma atividade renovada. Mais significativo foi o trabalho de J. J. Gibson (1950, 1979), que assinalou a importância do fluxo óptico, bem como dos gradientes de textura na avaliação de variáveis ambientais como inclinação e declividade de superfícies. Ele voltou a enfatizar a importância do estímulo e o quanto esse estímulo era rico. Gibson enfatizou o papel do observador ativo cujo movimento autodirigido facilita a captação de informação sobre o ambiente externo.

A visão computacional foi fundada na década de 1960. A tese de Roberts (1963) no MIT foi uma das primeiras publicações no campo, introduzindo ideias-chave como detecção de aresta e correspondência baseada em modelo. Existe uma lenda urbana de que Marvin Minsky atribuiu o problema de "resolver" a visão por computador para um aluno de pós-graduação como um projeto de verão. Segundo Minsky, a lenda é falsa — na verdade, foi um estudante de graduação. Mas foi um estudante excepcional, Gerald Jay Sussman (que agora é professor no MIT), e a tarefa não era "resolver" a visão, mas investigar alguns aspectos dela.

Nas décadas de 1960 e 1970, o progresso era lento, prejudicado consideravelmente pela falta de recursos computacionais e de armazenamento. O nível baixo de processamento visual recebia muita atenção. A técnica Canny, amplamente utilizada, de detecção de arestas, foi introduzida em Canny (1986). Técnicas para encontrar os limites de textura com base em multiescala, filtragem de imagens por multiorientação, pertencem ao período de trabalho como de Malik e Perona (1990). A combinação de múltiplas pistas — brilho, textura e cores — para encontrar as curvas de contorno em uma estrutura de aprendizado foi mostrada por Martin, Fowlkes e Malik (2004) para melhorar consideravelmente o desempenho.

O problema intimamente relacionado de encontrar regiões de brilho, cor e textura coerentes, naturalmente presta-se às formulações em que encontrar a melhor partição torna-se um problema de otimização. Três exemplos principais foram a abordagem de Geman e Geman (1984), de Campos de Markov Aleatórios, a formulação variacional de Mumford e Shah (1989) e cortes normalizados por Shi e Malik (2000).

Durante grande parte das décadas de 1960, 1970 e 1980, havia dois paradigmas distintos em que o reconhecimento visual foi perseguido, ditado por perspectivas diferentes sobre o que era percebido

como sendo o problema principal. A pesquisa de visão por computador sobre o reconhecimento de objetos em grande parte focou questões decorrentes da projeção de objetos tridimensionais em imagens bidimensionais. A ideia de alinhamento também foi primeiro introduzida por Roberts, ressurgiu na década de 1980 no trabalho de Lowe (1987) e Huttenlocher e Ullman (1990). Também foi popular uma abordagem com base na descrição de formas em termos volumétricos primitivos, com **cilindros generalizados**, introduzida por Tom Binford (1971).

Em contraste, a comunidade de reconhecimento de padrões via os aspectos do problema de 3-D para 2-D como não significativos. Seus exemplos motivadores eram em domínios como o reconhecimento de caracteres ópticos e de código postal manuscrito, onde a principal preocupação era aprender as variações típicas características de uma classe de objetos e separá-los das outras classes. Veja LeCun *et al*. (1995) para uma comparação das abordagens.

No final dos anos 1990, esses dois paradigmas começaram a convergir, à medida que ambos os lados adotaram a modelagem probabilística e técnicas de aprendizagem que foram se tornando populares em toda a IA. Duas linhas de trabalho contribuíram de forma significativa. Uma delas foi a pesquisa sobre detecção de rosto, tal como a de Rowley, Baluja e Kanade (1996) e de Viola e Jones (2002b), que demonstraram o poder das técnicas de reconhecimento de padrões em tarefas claramente importantes e úteis. Outra foi o desenvolvimento de descritores de ponto, que permitem construir vetores de características de partes de objetos. Foi iniciado por Schmid e Mohr (1996). O descritor SIFT de Lowe (2004) é amplamente utilizado. O descritor HOG é devido a Dalal e Triggs (2005).

Ullman (1979) e Longuet-Higgins (1981) foram trabalhos anteriores influentes de reconstrução a partir de várias imagens. As preocupações sobre a estabilidade da estrutura do movimento foram significativamente dissipadas pelo trabalho de Tomasi e Kanade (1992), que mostrou que, com o uso de vários quadros, a forma pode ser recuperada com bastante precisão. Na década de 1990, com grande aumento em velocidade e capacidade de armazenamento do computador, a análise do movimento encontrou muitas aplicações novas. A construção de modelos geométricos de cenas do mundo real para execução por técnicas de computação gráfica provou ser particularmente popular, liderada pelos algoritmos de reconstrução, como o desenvolvido por Debevec, Taylor e Malik (1996). Os livros de Hartley e Zisserman (2000) e Faugeras *et al*. (2001) fornecem um tratamento abrangente da geometria de visões múltiplas.

Para imagens individuais, a inferência da forma a partir do sombreamento foi inicialmente estudada por Horn (1970), e Horn e Brooks (1989) apresentam uma pesquisa extensiva das teses principais de um período em que esse era um problema muito estudado. Gibson (1950) foi o primeiro a propor gradientes de textura como sugestão para dar forma, embora uma análise global para superfícies curvas apareça pela primeira vez em Garding (1992) e Malik e Rosenholtz (1997). A matemática dos contornos de oclusão e a compreensão de forma mais geral dos acontecimentos visuais na projeção de objetos de curva suave deve muito ao trabalho de Koenderink e Van Doorn, que encontrou um tratamento extenso na *Solid Shape* de Koenderink (1990). Nos últimos anos, a atenção voltou-se para tratar o problema da recuperação da forma e superfície a partir de uma única imagem como um problema de inferência probabilística, onde sinais geométricos não são modelados explicitamente, mas utilizados implicitamente em uma estrutura de aprendizagem. Um bom representante é o trabalho de Hoiem, Efros e Hebert (2008).

Para o leitor interessado na visão humana, Palmer (1999) fornece o tratamento mais abrangente; Bruce *et al*. (2003) é um livro mais curto. Os livros de Hubel (1988) e Rock (1984) são introduções amigáveis centralizadas em neurofisiologia e percepção, respectivamente. O livro *Vision* de David Marr (Marr, 1982) desempenhou um papel histórico na conexão da visão de computador de psicofísica e neurobiologia. Enquanto muitos de seus modelos específicos não resistiram ao teste do tempo, a perspectiva teórica a partir da qual cada tarefa é analisada em um informativo computacional e em nível de implementação ainda é esclarecedora.

Para a visão por computador, o livro mais abrangente é Forsyth e Ponce (2002). Trucco e Verri (1998) tem menor importância. Horn (1986) e Faugeras (1993) são dois livros mais antigos e ainda úteis.

As principais revistas de visão por computador são IEEE *Transactions on Pattern Analysis and Machine Intelligence and International Journal of Computer Vision.* As conferências sobre visão por computador incluem ICCV (International Conference on Computer Vision), CVPR (Computer Vision and Pattern Recognition) e ECCV (European Conference on Computer Vision). A pesquisa com componente de aprendizagem de máquina também foi publicada na conferência NIPS (Neural Information Processing Systems), e o trabalho sobre a interface com gráficos de computador muitas vezes aparece na conferência ACM SIGGRAPH (Special Interest Group in Graphics).

EXERCÍCIOS

24.1 Na sombra de uma árvore com copa densa e frondosa, vemos vários pontos de luz. Surpreendentemente, todos eles parecem circulares. Por quê? Afinal, os intervalos entre as folhas através dos quais o Sol brilha provavelmente não têm forma circular.

24.2 Considere o quadro de uma esfera branca flutuando em frente a um pano de fundo preto. A curva da imagem que separa os pixels é algo chamado “contorno” da esfera. Mostre que o contorno de uma esfera, visto sob a perspectiva de uma câmera, pode ser uma elipse. Por que as esferas não parecem como elipses para você?

24.3 Considere um cilindro infinitamente longo de raio r, orientado com seu eixo ao longo do eixo y. O cilindro tem superfície lambertiana e é visualizado por uma câmera ao longo do semieixo z positivo. O que você espera ver na imagem se o cilindro for iluminado por uma fonte pontual a uma distância infinita, posicionada no semieixo x positivo? Explique sua resposta desenhando os contornos de brilho equivalente na imagem projetada. Os contornos de brilho equivalente estão uniformemente espaçados?

24.4 As arestas em uma imagem podem corresponder a uma variedade de eventos em uma cena. Considere a capa deste livro e suponha que ela fosse um quadro de uma cena tridimensional real. Identifique dez arestas de brilho diferente na imagem e, para cada uma, diga se ela corresponde a uma descontinuidade de (a) profundidade, (b) normal à superfície, (c) refletância ou (d) iluminação.

24.5 Um sistema estereoscópico está sendo contemplado para mapeamento de um terreno. Ele consistirá em duas câmeras CCD, cada uma com 512 × 512 pixels em um sensor quadrado de 10 cm × 10 cm. As lentes a serem utilizadas têm a distância focal de 16 cm, com o foco fixo no infinito.

Para pontos correspondentes (u_1, v_1) na imagem da esquerda e (u_2, v_2) na imagem da direita, $v_1 = v_2$ porque os eixos x nos dois planos das imagens são paralelos às linhas epipolares. Os eixos ópticos das duas câmeras são paralelos. A linha de base entre as câmeras tem 1 metro.

a. Se o intervalo mais próximo a ser medido é de 16 metros, qual será a maior disparidade que ocorrerá (em pixels)?

b. Qual é a resolução do intervalo a 16 metros, devido ao espaçamento de pixels?

c. Que intervalo corresponde a uma disparidade de um pixel?

24.6 Quais das afirmativas a seguir são verdadeiras e quais são falsas?

a. Encontrar pontos correspondentes em imagens estereoscópicas é a fase mais fácil do processo de descoberta da profundidade estereoscópica.

b. A descoberta da forma a partir da textura pode ser realizada projetando-se uma grade de faixas de luz sobre a cena.

c. Linhas com comprimentos iguais na cena sempre se projetam em comprimentos iguais na imagem.

d. Linhas retas na imagem necessariamente correspondem a linhas retas na cena.

24.7 (Cortesia de Pietro Perona.) A Figura 24.27 mostra duas câmeras em X e Y observando uma cena. Desenhe a imagem vista em cada câmera supondo que todos os pontos nomeados estejam no mesmo plano horizontal. O que podemos concluir a partir dessas duas imagens sobre as distâncias relativas dos pontos A, B, C, D e E em relação à linha de base da câmera? Em que se baseia sua conclusão?

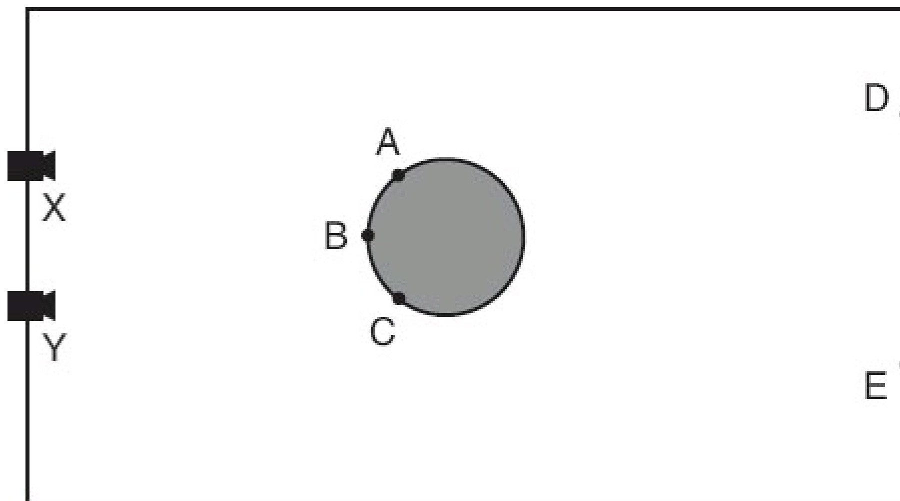


Figura 24.27 Vista superior de um sistema de visão de duas câmeras observando uma garrafa com uma parede por trás dela.

CAPÍTULO 25

Robótica

Em que os agentes são dotados de efetuadores físicos com os quais podem fazer das suas.

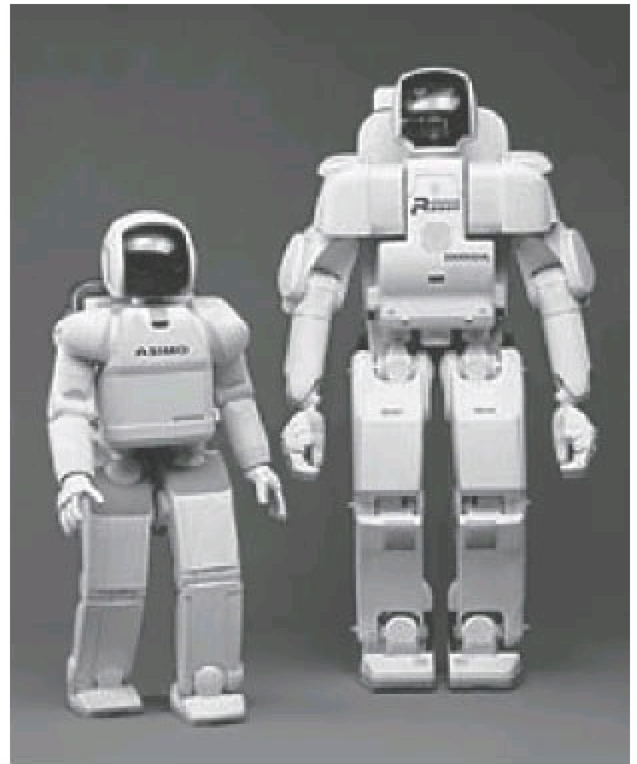
25.1 INTRODUÇÃO

Os **robôs** são agentes físicos que executam tarefas manipulando o mundo físico. Para isso, eles são equipados com **efetuadores** como pernas, rodas, articulações e garras. Os efetuadores têm um único propósito: exercer forças físicas sobre o ambiente.[1] Os robôs também estão equipados com **sensores**, que lhes permitem perceber seu ambiente. A robótica atual emprega um conjunto diversificado de sensores, inclusive câmeras e ultrassom para medir o ambiente, além de giroscópios e acelerômetros para medir o movimento do próprio robô.

A maior parte dos robôs atuais se enquadra em uma dentre três categorias principais. Os **manipuladores**, ou braços robôs (Figura 25.1(a)), estão fisicamente ancorados (ou fixos) ao seu local de trabalho, por exemplo, em uma linha de montagem industrial ou na estação espacial internacional. O movimento do manipulador em geral envolve uma cadeia inteira de articulações controláveis, permitindo que esses robôs coloquem seus efetuadores em qualquer posição dentro do local de trabalho. Sem dúvida, os manipuladores são o tipo mais comum de robôs industriais, com mais de um milhão de unidades instaladas em todo o mundo. Alguns manipuladores móveis são usados em hospitais para auxiliar os cirurgiões. Poucos fabricantes de automóveis poderiam sobreviver sem manipuladores robóticos, e alguns manipuladores foram usados até mesmo para gerar trabalhos artísticos originais.



(a)



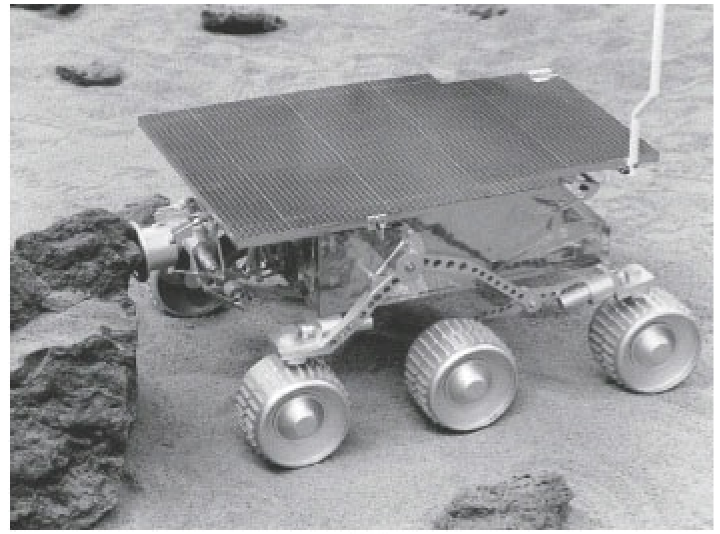
(b)

Figura 25.1 (a) Manipulador robótico industrial para empilhar sacos sobre um estrado. Imagem cedida por Nachi Robotic Systems. (b) Os robôs humanoides P3 e Asimo da Honda.

A segunda categoria é o **robô móvel**. Os robôs móveis se deslocam por seu ambiente usando rodas, pernas ou mecanismos semelhantes. Eles foram projetados para uso na entrega de alimentos em hospitais, para mover contêineres em docas de carga e tarefas semelhantes. **Veículos terrestres não tripulados**, ou UGVs, se conduzem de forma autônoma em ruas, rodovias e *off-road*. O **veículo planetário** mostrado na Figura 25.2(b) explorou Marte por um período de três meses em 1997. Os robôs subsequentes da Nasa incluem os gêmeos Mars Exploration Rovers (um é retratado na capa deste livro), que aterrissaram em 2003 e ainda estavam operando seis anos depois. Outros tipos de robôs móveis incluem **veículos aéreos não tripulados** (UAV — *unmanned air vehicle*), comumente usados para a vigilância, pulverizações e operações militares. A Figura 25.2(a) mostra um UAV utilizado comumente pelos militares dos Estados Unidos. Veículos autônomos subaquáticos (AUVs — ***autonomous underwater vehicles***) são usados em exploração no fundo do mar. Os robôs móveis entregam pacotes no local de trabalho e aspiram o piso em casa.



(a)



(b)

Figura 25.2 (a) Predador, um veículo aéreo não tripulado (UAV) utilizado pelo exército dos Estados Unidos. Imagem cedida por General Atomics Aeronautical Systems. (b) Sojourner da Nasa, um robô móvel que explorou a superfície de Marte em julho de 1997.

O terceiro tipo de robô combina mobilidade com manipulação e é muitas vezes chamado de **manipulador móvel**. **Robôs humanoides** imitam o torso humano. A Figura 25.1(b) mostra dois desses robôs humanoides, ambos fabricados pela Honda Corp. no Japão. Manipuladores móveis podem aplicar seus efetuadores adicionais em campos mais amplos que os manipuladores fixos, mas sua tarefa se torna mais difícil porque eles não têm a rigidez que o ponto de fixação oferece.

O campo da robótica também inclui dispositivos protéticos (membros artificiais, orelhas e olhos para seres humanos), ambientes inteligentes (como a casa inteira que é equipada com sensores e efetuadores) e sistemas com vários corpos, nos quais a ação robótica é alcançada por enxames de pequenos robôs cooperativos.

Os robôs reais em geral devem lidar com ambientes parcialmente observáveis, estocásticos, dinâmicos e contínuos. Muitos ambientes de robôs são sequenciais e também de multiagentes. A observabilidade parcial e o caráter estocástico são o resultado do trabalho em um mundo grande e complexo. O robô câmera não pode ver em torno de obstáculos, e os comandos de movimentação estão sujeitos à incerteza, devido ao deslizamento de engrenagens, à fricção etc. Além disso, o mundo real se recusa obstinadamente a operar mais rápido que o tempo real. Em um ambiente simulado, é possível usar algoritmos simples (como o algoritmo de aprendizado Q descrito no Capítulo 21) para aprender em algumas horas de CPU a partir de milhões de tentativas. Em um ambiente real, se poderia levar anos para executar essas tentativas. Além disso, as colisões reais de fato danificam, ao contrário das colisões simuladas. Os sistemas robóticos práticos precisam incorporar o conhecimento anterior sobre o robô, seu ambiente físico e as tarefas que o robô executará, de forma que o robô possa aprender com rapidez e executar suas tarefas em segurança.

A robótica reúne muitos dos conceitos que vimos no início do livro, incluindo estimação de estado probabilístico, percepção, planejamento, aprendizagem não supervisionada e reforço de aprendizagem. Para alguns desses conceitos serve como exemplo de aplicação desafiador. Para outros conceitos este capítulo inova ao introduzir a versão contínua de técnicas que vimos anteriormente apenas no caso discreto.

25.2 HARDWARE DE ROBÔS

Até agora neste livro, assumimos a arquitetura do agente — sensores, efetuadores e processadores — como dada e nos concentramos no programa do agente. O sucesso dos robôs reais também depende bastante do projeto de sensores e efetuadores apropriados para a tarefa.

25.2.1 Sensores

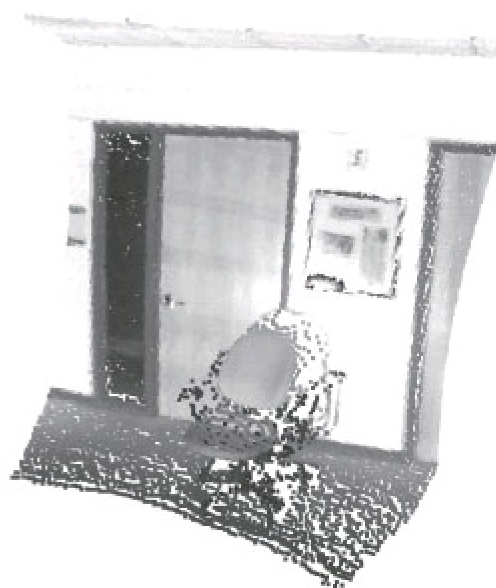
Os sensores são a interface perceptiva entre robôs e seus ambientes. Os **sensores passivos**, como câmeras, são verdadeiros observadores do ambiente: eles captam sinais gerados por outras fontes no ambiente. Os **sensores ativos**, como o sonar, enviam energia ao ambiente. Eles contam com o fato de que essa energia é refletida de volta para o sensor. Os sensores ativos tendem a fornecer mais informações que os sensores passivos, mas ao custo de maior consumo de energia e com perigo de interferência quando vários sensores ativos são usados ao mesmo tempo. Quer sejam ativos ou passivos, os sensores podem ser divididos em três tipos, dependendo do fato de registrarem distâncias até objetos, imagens inteiras do ambiente ou propriedades do próprio robô.

Muitos robôs móveis fazem uso de **telêmetros**, que são sensores que medem a distância até objetos próximos. No princípio da robótica, normalmente, os robôs eram equipados com **sensores de sonar**. Os sensores de sonar emitem ondas sonoras direcionais, que são refletidas por objetos, com uma parte do som voltando ao sensor. Desse modo, o tempo e a intensidade desse sinal de retorno transmitem informações sobre a distância até objetos próximos. O sonar é a tecnologia de escolha para veículos subaquáticos autônomos. A **visão estéreo** (veja a Seção 24.4.2) baseia-se em múltiplas câmeras para retratar o ambiente sob pontos de vista ligeiramente diferentes, analisando a paralaxe resultante dessas imagens para calcular a amplitude dos objetos que rodeiam. Para robôs móveis terrestres, o sonar e a visão estéreo são utilizados agora raramente porque não apresentam precisão confiável.

Agora, a maioria dos robôs terrestres está equipada com telêmetros ópticos. Como sensores de sonar, sensores de alcance óptico emitem sinais ativos (luz) e medem o tempo até que um reflexo desse sinal chegue de volta ao sensor. A Figura 25.3(a) mostra uma **câmera do tempo de voo.** Essa câmera adquire amplitude de imagem, como mostrado na Figura 25.3 (b), em até 60 frames por segundo. Outros sensores de alcance utilizam feixes de *laser* e câmeras especiais de 1 pixel que podem ser direcionadas utilizando arranjos complexos de espelhos ou elementos rotativos. Esses sensores são chamados de ***scanning lidars*** (abreviação de *light detection and ranging*). *Scanning lidars* tendem a fornecer maior alcance que as câmeras de tempo de voo e a ter desempenho melhor na luz do dia.



(a)



(b)

Figura 25.3 (a) Câmera do tempo de voo; imagem cedida pela Mesa Imaging GmbH. (b) Amplitude da imagem em 3-D obtida com essa câmera. A amplitude da imagem torna possível detectar obstáculos e objetos nas imediações de um robô.

Outros sensores de alcance comum incluem o radar, que muitas vezes é o sensor de escolha para UAVs. Os sensores de radar podem medir distâncias de vários quilômetros. Em outro extremo, o fim dos sensores de alcance são os **sensores táteis**, como *whiskers*, *bamp panels* e sensíveis ao toque da pele. Esses sensores medem a amplitude baseados em contato físico e podem ser empregados apenas para objetos de detecção muito próximos ao robô.

Uma segunda classe importante de sensores são os **sensores de localização**. A maioria deles utiliza o sensor de amplitude como componente principal para determinar a localização. Ao ar livre, o **sistema de posicionamento global** (GPS) é a solução mais comum para o problema de localização. O GPS mede a distância até os satélites que emitem sinais pulsados. Atualmente, existem 31 satélites em órbita, transmitindo sinais em várias frequências. Os receptores de GPS podem recuperar a distância até esses satélites através da análise de mudanças de fase. Por triangulação de sinais de vários satélites, os receptores de GPS podem determinar sua localização absoluta na Terra em poucos metros. O **GPS diferencial** envolve um segundo receptor no solo com localização conhecida, proporcionando precisão milimétrica em condições ideais. Infelizmente, o GPS não funciona no interior ou debaixo d'água. A localização no interior, muitas vezes, é alcançada anexando balizas no ambiente em locais conhecidos. Muitos ambientes internos estão repletos de estações de base sem fio, que podem ajudar os robôs a localizar através da análise do sinal sem fio. Sob a água, as balizas sonares ativas podem fornecer o sentido de localização, utilizando o som para informar AUVs de suas distâncias relativas até essas balizas.

A terceira classe importante é a dos **sensores proprioceptivos**, que informam ao robô seu próprio estado. Para medir a configuração exata de uma articulação robótica, os motores frequentemente são equipados com **decodificadores de eixos** que contam a revolução de motores em pequenos incrementos. Em braços de robôs, os decodificadores de eixos podem fornecer informações exatas ao longo de qualquer período de tempo. Em robôs móveis, os decodificadores de eixos que informam as revoluções das rodas podem ser usados para **odometria** — a medição da distância percorrida.

Infelizmente, as rodas tendem a travar e patinar, e portanto a odometria só é exata quando se consideram distâncias curtas. Forças externas, como a corrente para AUVs e o vento para UAVs, aumentam a incerteza posicional. Os **sensores inerciais**, como os giroscópios, podem ajudar, mas não são capazes de impedir sozinhos a acumulação inevitável de incerteza de posição.

Outros aspectos importantes do estado do robô são medidos pelos **sensores de força** e pelos **sensores de torque**. Esses sensores são indispensáveis quando os robôs manipulam objetos frágeis ou objetos cuja forma e posição exatas são desconhecidas. Imagine um manipulador robótico de uma tonelada trocando uma lâmpada incandescente. Seria muito fácil aplicar uma força excessiva e quebrar a lâmpada. Os sensores de força permitem ao robô perceber a dificuldade para agarrar a lâmpada, e os sensores de torque permitem que ele detecte a dificuldade de girá-la. Os bons sensores podem medir forças em três direções de translação e em três direções de rotação. Eles fazem isso em uma frequência de várias centenas de vezes por segundo, para que um robô possa detectar rapidamente forças inesperadas e corrigir suas ações antes de quebrar uma lâmpada.

25.2.2 Efetuadores

Os efetuadores são os meios pelos quais os robôs se movem e alteram a forma de seus corpos. Para entender o projeto de efetuadores, será útil mencionar primeiro o movimento e a forma em seus aspectos abstratos utilizando o conceito de **grau de liberdade** (GDL). Contamos um grau da liberdade para cada direção independente em que um robô ou um de seus efetuadores pode se mover. Por exemplo, um robô rígido de movimentos livres como um AUV tem seis graus da liberdade, três para sua posição (x, y, z) no espaço e três para sua orientação angular, conhecidos como *yaw*, *roll* e *pitch*. Esses seis graus definem o **estado cinemático**[2] ou **pose** do robô. O **estado dinâmico** de um robô inclui uma dimensão adicional para a taxa de mudança de cada dimensão cinemática, isto é, suas velocidades.

Para corpos não rígidos, existem graus adicionais de liberdade dentro do próprio robô. Por exemplo, o cotovelo de um braço humano possui dois graus de liberdade. Ele pode flexionar a parte superior na direção ou para longe e rodar para a direita ou esquerda. O pulso tem três graus de liberdade. Pode se mover para cima e para baixo, lado a lado e também pode girar. As articulações dos robôs também têm um, dois ou três graus de liberdade cada uma. São necessários seis graus da liberdade para colocar um objeto, como uma mão, em um ponto específico em uma orientação específica. O braço da Figura 25.4(a) tem exatamente seis graus da liberdade, criados por cinco **articulações de revolução**, que geram o movimento de rotação, e uma **articulação prismática**, que gera um movimento de deslizamento. Você pode verificar que o braço humano como um todo tem mais de seis graus de liberdade, realizando um experimento simples: ponha sua mão sobre a mesa e note que você ainda tem a liberdade de girar seu cotovelo sem mudar a configuração da mão. Os manipuladores que têm mais graus de liberdade que o necessário para colocar um efetuador final em uma posição de destino são mais fáceis de controlar que os robôs com apenas o número mínimo de GDLs. Portanto, muitos manipuladores industriais têm sete GDLs e não seis.

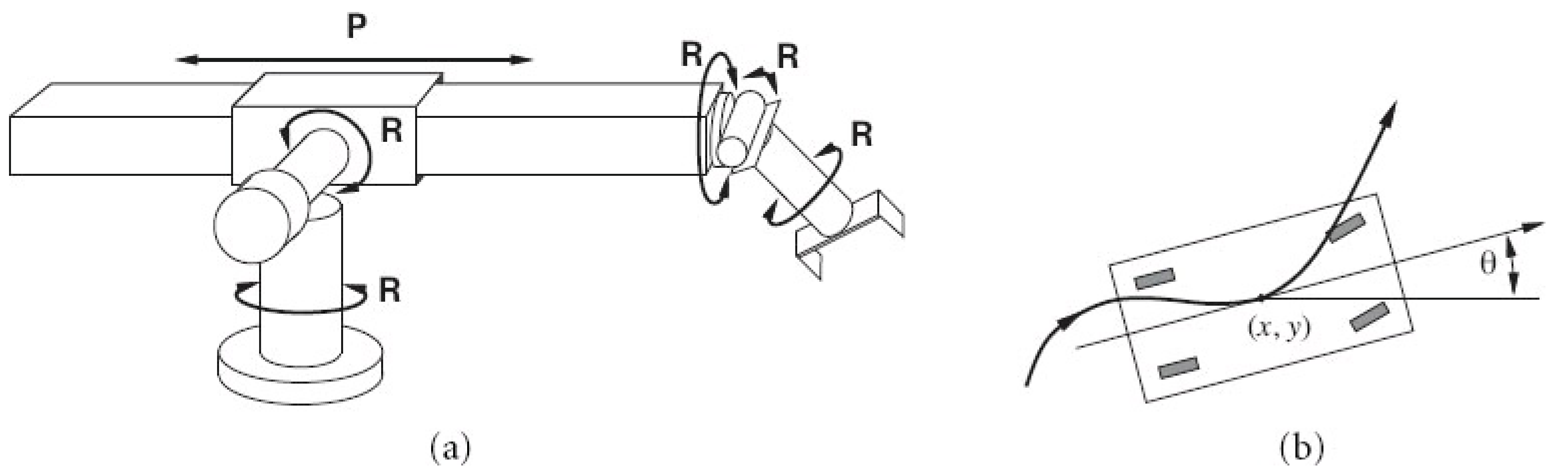


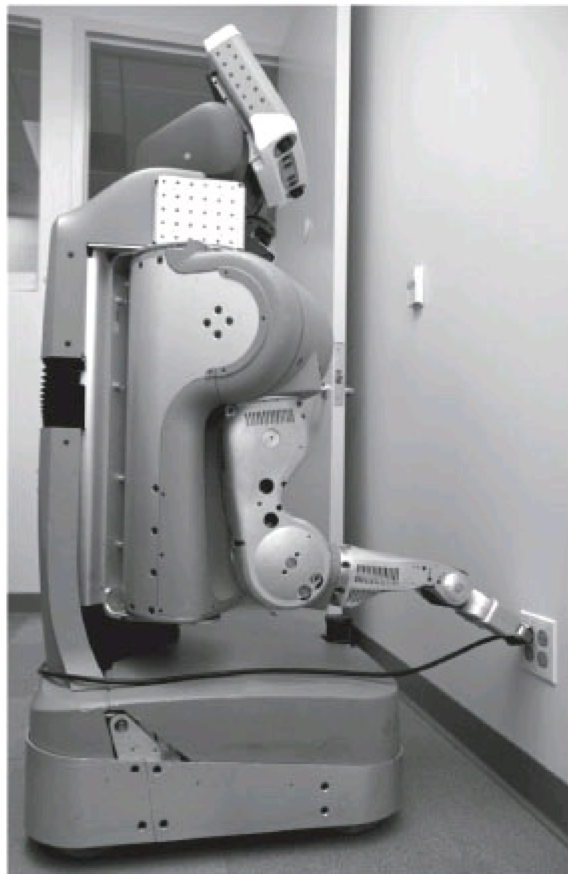
Figura 25.4 (a) O manipulador de Stanford, um antigo braço robô com cinco articulações de revolução (R) e uma articulação prismática (P), perfazendo um total de seis graus de liberdade. (b) Movimento de um veículo não holonômico de quatro rodas com controle de direção nas rodas dianteiras.

Para robôs móveis, os GDLs não são necessariamente iguais ao número de elementos movidos. Por exemplo, considere um carro médio: ele pode se mover para a frente ou para trás e também pode virar à esquerda ou à direita, o que lhe dá dois GDLs. Em contraste, uma configuração cinemática de um carro é tridimensional: em uma superfície plana aberta, é possível manobrar facilmente um carro para qualquer ponto (x, y), em qualquer orientação (veja a Figura 25.4(b)). Desse modo, o carro tem três **graus efetivos de liberdade**, mas dois **graus controláveis de liberdade**. Dizemos que um robô é **não holonômico** se ele tem mais GDLs efetivos que GDLs controláveis, e **holonômico** se os dois números são iguais. Os robôs holonômicos são mais fáceis de controlar — seria muito mais fácil estacionar um automóvel que pudesse se mover para os lados e também para a frente e para trás —, mas os robôs holonômicos também são mecanicamente mais complexos. Em sua maioria, os braços robôs são holonômicos, e a maioria dos robôs móveis é não holonômica.

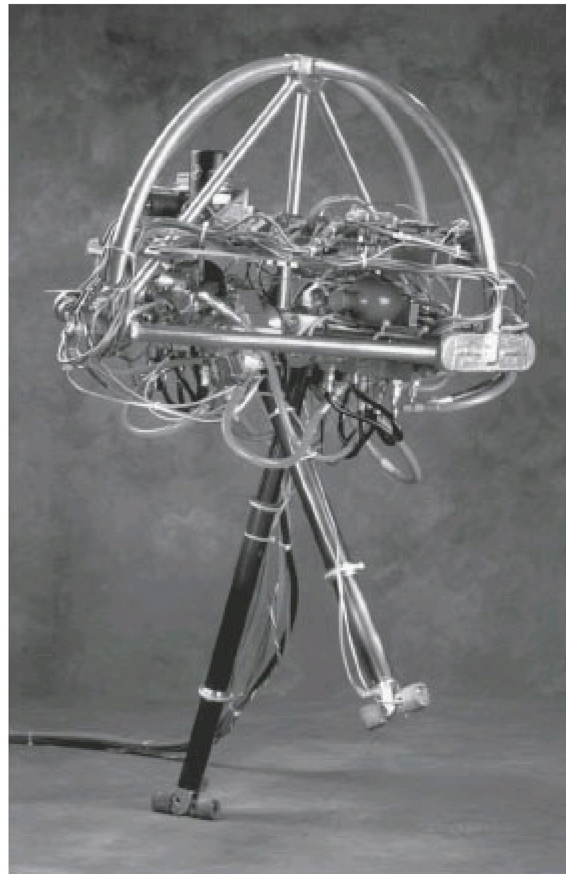
Para robôs móveis, existe uma variedade de mecanismos para locomoção, incluindo rodas, esteiras e pernas. Os robôs de **tração diferencial** possuem duas rodas acionadas independentemente (ou esteiras), uma em cada lado, como em um tanque militar. Se ambas as rodas se moverem na mesma velocidade, o robô se moverá em linha reta.

Se elas se moverem em direções opostas, o robô vai girar sobre o ponto. Uma alternativa é a **tração sincronizada**, na qual cada roda pode se mover e girar em torno de seu próprio eixo. Para evitar o caos, as rodas são bem coordenadas. Quando se deslocam em linha reta, por exemplo, todas as rodas apontam na mesma direção e se movem na mesma velocidade. Tanto a tração diferencial quanto a tração sincronizada são holonômicas. Alguns robôs mais dispendiosos utilizam transmissões holonômicas que, em geral, envolvem três ou mais rodas que podem ser orientadas ou movidas independentemente.

Alguns robôs móveis possuem braços. A Figura 25.5(a) mostra um robô com dois braços. Os braços do robô utilizam molas para compensar a gravidade, e elas proporcionam resistência mínima às forças externas. Tal projeto minimiza o perigo físico para as pessoas que podem tropeçar em um robô. Essa é uma consideração-chave na implantação de robôs em ambientes domésticos.



(a)



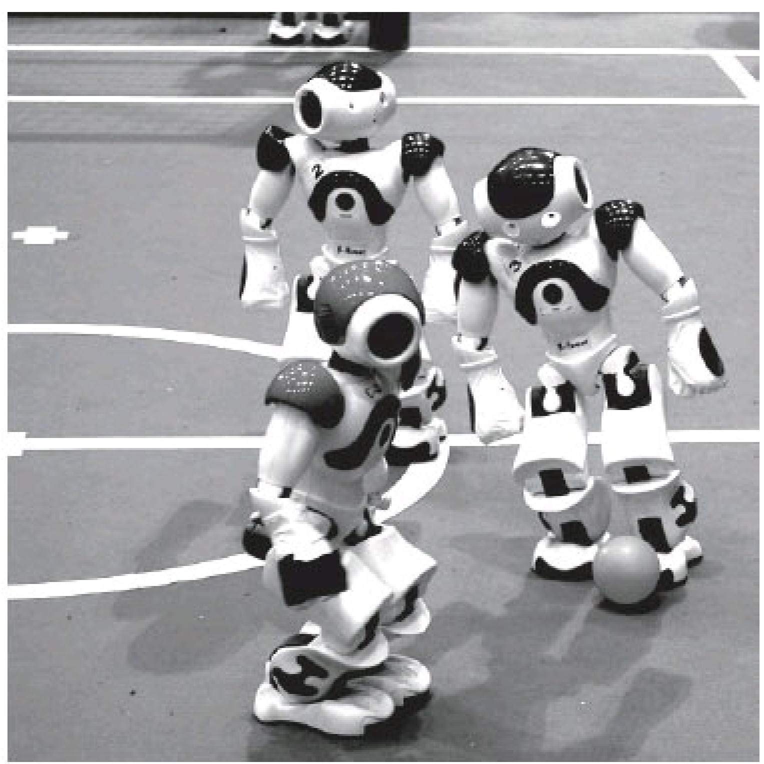
(b)

Figura 25.5 (a) Manipulador móvel com seu cabo de carga conectado em uma tomada de parede. Imagem cedida por Willow Garage, © 2009. (b) Um dos robôs com pernas Marc Raibert em movimento.

As pernas, diferentemente das rodas, podem percorrer terrenos muito acidentados. Porém, é notório que as pernas são lentas em superfícies planas, e sua construção é mecanicamente difícil. Os pesquisadores de robótica experimentaram projetos que variavam desde uma única perna até dezenas de pernas. Os robôs providos de pernas foram feitos para caminhar, correr e até mesmo saltar, como vemos no caso do robô provido de pernas na Figura 25.5(a). Esse robô é **dinamicamente estável**, o que significa que ele pode permanecer de pé enquanto salta. Um robô que pode permanecer em equilíbrio sem movimentar as pernas é chamado **estaticamente estável**. Um robô é estaticamente estável se seu centro de gravidade está acima do polígono formado por suas pernas. O robô (de quatro pernas) quadrúpede mostrado na Figura 25.6(a) pode aparecer estaticamente estável. No entanto, ele anda levantando várias pernas ao mesmo tempo, o que o torna dinamicamente estável. O robô pode andar na neve e no gelo, e não cai mesmo se for chutado (como se demonstra em vídeos on-line disponíveis). Os robôs de duas pernas, como os da Figura 25.6 (b), são dinamicamente estáveis.



(a)



(b)

Figura 25.6 (a) O robô "Big Dog" com quatro pernas e dinamicamente estável. Imagem cedida por Boston Dynamics, © 2009. (b) Competição de 2009 da RoboCup Standard Platform League, mostrando a equipe vencedora, B-Human, do centro DFKI da Universidade de Bremen. Em toda a partida superaram seus adversários por 64:1. Seu sucesso foi construído sobre estimação de estado probabilístico utilizando filtros de partículas e filtros de Kalman; sobre modelos de aprendizagem de máquina para otimização de marcha; e sobre dinâmica dos movimentos de chute. Imagem cedida por DFKI, © 2009.

Há possibilidade de outros métodos de movimento: veículos aéreos utilizam hélices ou turbinas; veículos subaquáticos utilizam hélices ou propulsores, semelhantes aos utilizados em submarinos. Dirigíveis robóticos contam com efeito térmico para manter-se no alto.

Sozinhos, os sensores e os efetuadores não fazem um robô. Um robô completo também precisa de uma fonte de energia para alimentar seus efetuadores. O **motor elétrico** é o mecanismo mais popular, tanto para atuação dos manipuladores quanto para locomoção, mas a **atuação pneumática** que utiliza gás comprimido e a **atuação hidráulica** que utiliza fluidos pressurizados também têm seus nichos de aplicação.

25.3 PERCEPÇÃO ROBÓTICA

A percepção é o processo pelo qual os robôs mapeiam medições de sensores em representações internas do ambiente. A percepção é difícil porque, em geral, os sensores são ruidosos e o ambiente é parcialmente observável, imprevisível e frequentemente dinâmico. Em outras palavras, os robôs têm todos os problemas de **estimativa de estado** (ou **filtragem**) que discutimos na Seção 15.2. Como regra prática, as boas representações internas têm três propriedades: elas contêm informações suficientes para o robô tomar as decisões corretas, são estruturadas de tal modo que possam ser

atualizadas com eficiência e são naturais, no sentido de que as variáveis internas correspondem a variáveis de estados naturais no mundo físico.

No Capítulo 15, vimos que os filtros de Kalman, os MOMs e as redes bayesianas dinâmicas podem representar os modelos de transição e de sensores de um ambiente parcialmente observável, e descrevemos algoritmos exatos e aproximados para atualizar o **estado de crença** — a distribuição de probabilidade posterior sobre as variáveis de estados do ambiente. Vários modelos de redes bayesianas dinâmicas para esse processo foram mostrados no Capítulo 15. Para problemas de robótica, normalmente incluímos as ações passadas do próprio robô como variáveis observadas no modelo. A Figura 25.7 mostra a notação usada neste capítulo: $\mathbf{X}_t$ é o estado do ambiente (incluindo o robô) no instante t, $\mathbf{Z}_t$ é a observação recebida no instante t e A_t é a ação executada depois que a observação é recebida.

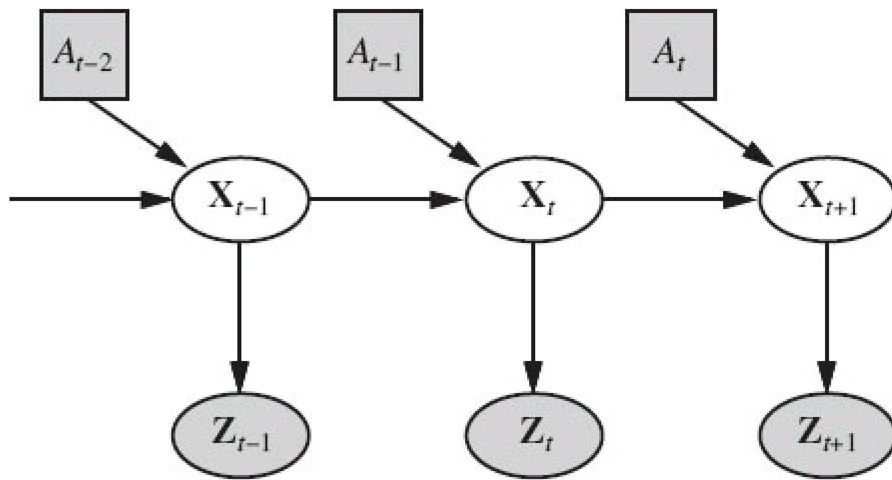


Figura 25.7 A percepção de robôs pode ser visualizada como inferência temporal a partir de sequências de ações e medições, como ilustra essa rede bayesiana dinâmica.

Gostaríamos de calcular o novo estado de crença, $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t})$, a partir do estado de crença atual $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ e da nova observação $\mathbf{z}_{t+1}$. Fizemos isso na Seção 15.2, mas aqui há duas diferenças: definimos as condições explicitamente sobre as ações como também sobre as observações, e agora temos de lidar com variáveis *contínuas*, e não com variáveis *discretas*. Desse modo, modificamos a equação de filtragem recursiva (15.5) para usar a integração em lugar do somatório:

$$\begin{aligned}&\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t})\\&= \alpha \mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t)\, P(\mathbf{x}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})\, d\mathbf{x}_t \,.\end{aligned} \tag{25.1}$$

A equação afirma que a distribuição posterior sobre as variáveis de estados $\mathbf{X}$ no instante $t + 1$ é calculada recursivamente a partir da estimativa correspondente em um período de tempo anterior. Esse cálculo envolve a ação anterior a_t e a medição atual do sensor $\mathbf{z}_{t+1}$. Por exemplo, se nossa meta é desenvolver um robô para jogar futebol, $\mathbf{X}_{t+1}$ pode ser a posição da bola de futebol em relação ao robô. A distribuição posterior $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ é uma distribuição de probabilidade sobre todos os estados, que capta o que sabemos das medições de sensores e de controles passados. A Equação

25.1 nos diz como avaliar recursivamente essa posição reunindo de forma incremental medições de sensores (por exemplo, imagens de câmeras) e comandos de movimentação do robô. A probabilidade $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t)$ é chamada **modelo de transição** ou **modelo de movimento** e $\mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1})$ se denomina **modelo de sensores**.

25.3.1 Localização e mapeamento

A **localização** é o problema de descobrir onde os objetos estão — incluindo o próprio robô. O conhecimento sobre onde estão é o cerne de qualquer interação física bem-sucedida com o meio ambiente. Por exemplo, robôs manipuladores devem saber a localização de objetos que procuram para manipular; robôs que navegam devem saber onde estão para encontrar o seu caminho de volta.

Para manter a simplicidade, vamos considerar um robô móvel que se movimenta devagar em um plano do mundo 2-D. Vamos também assumir que o robô recebeu um mapa exato do ambiente (um exemplo de tal mapa aparece na Figura 25.10). A pose de um robô móvel desse tipo é definida por suas duas coordenadas cartesianas com valores x e y, e sua orientação é definida com o valor θ, como ilustra a Figura 25.8(a). (Note que excluímos as velocidades correspondentes e, assim, esse é um modelo cinemático em vez de um modelo dinâmico.) Se organizarmos esses três valores em um vetor, qualquer estado específico será dado por $\mathbf{X}_t = (x_t, y_t, \theta_t)^{\mathrm{T}}$. Até agora, tudo bem.

Na aproximação cinemática, cada ação consiste na especificação “instantânea” de duas velocidades — uma velocidade de translação v_t e uma velocidade de rotação ω_t. Para pequenos intervalos de tempo Δt, um modelo determinístico bruto do movimento de tais robôs é dado por:

$$\hat{\mathbf{X}}_{t+1} = f(\mathbf{X}_t, \underbrace{v_t, \omega_t}_{a_t}) = \mathbf{X}_t + \begin{pmatrix} v_t \Delta t \cos \theta_t \\ v_t \Delta t \sin \theta_t \\ \omega_t \Delta t \end{pmatrix} .$$

A notação $\hat{\mathbf{X}}$ se refere a uma previsão de estado determinística. É claro que os robôs físicos são um tanto imprevisíveis. Normalmente, isso é modelado por uma distribuição gaussiana com média $f(\mathbf{X}_t, v_t, \omega_t)$ e covariância Σ_x (veja no Apêndice A uma definição matemática).

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, v_t, \omega_t) \;=\; N(\hat{\mathbf{X}}_{t+1}, \mathbf{\Sigma}_x) .$$

Essa distribuição de probabilidade é o modelo de movimento do robô. Modela o efeito do movimento sobre a localização do robô.

Em seguida, precisamos de um modelo de sensores. Consideraremos dois tipos de modelos de sensores. O primeiro pressupõe que os sensores detectam características *estáveis* e *reconhecíveis* do ambiente, chamadas **marcos**. Para cada marco, são informados o intervalo e o porte. Suponha que o estado do robô seja $\mathbf{x}_t = (x_t, y_t, \theta_t)^{\mathrm{T}}$ e que ele detecte um marco cuja posição é dada por $(x_t, y_t)^{\mathrm{T}}$. Sem ruído, o intervalo e o porte podem ser calculados por simples geometria (veja a Figura 25.8(a)). A previsão exata do intervalo e do porte observados seria:

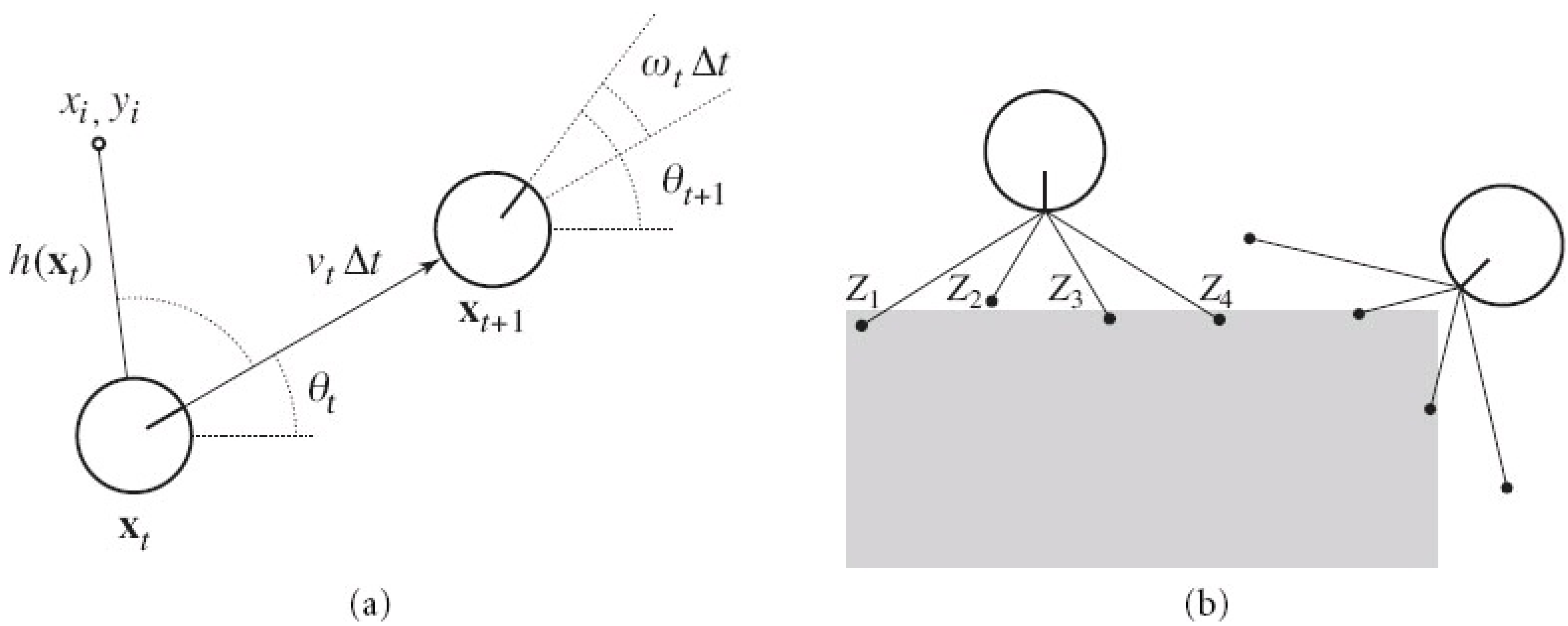


Figura 25.8 (a) Modelo cinemático simplificado de um robô móvel. O robô é apresentado como um círculo com uma marca mostrando a direção à frente dele. O estado $\mathbf{x}_t$ consiste da posição (x_t, y_t) (mostrada implicitamente) e da orientação θ_t. O estado novo $\mathbf{x}_{t+1}$ é obtido através de uma atualização na posição de $v_t\Delta_t$ e da orientação $\omega_t\Delta_t$. Também é exibido um marco em (x_i, y_i), observado no instante θ_t. (b) Modelo de sensor de varredura de intervalo. Duas poses possíveis do robô são mostradas para dada varredura de intervalo (z_1, z_2, z_3, z_4). É muito mais provável que a varredura de intervalo tenha sido gerada pela pose da esquerda do que pela pose da direita.

$$\hat{\mathbf{z}}_t = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan \frac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix}$$

Mais uma vez, o ruído distorce nossas medições. Para manter a simplicidade, poderíamos supor ruído gaussiano com covariância $\sum_Z$ fornecendo o modelo de sensor

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = N(\hat{\mathbf{z}}_t, \mathbf{\Sigma}_z) \; .$$

Com frequência, um modelo de sensores um pouco diferente é utilizado para um vetor de sensores de alcance, cada um dos quais tem uma relação fixa em relação ao robô. Tais sensores produzem um vetor de valores de intervalos $\mathbf{z}_t = (z_1,\ldots, z_M)^T$, cujos portes são fixos em relação ao robô. Dada uma pose $\mathbf{x}_t$, seja $\hat{z}_j$ o intervalo exato ao longo da j-ésima direção de feixe desde $\mathbf{x}_t$ até o obstáculo mais próximo. Como antes, isso será corrompido por ruído gaussiano. Em geral, supomos que os erros para as diferentes direções do feixe estão distribuídos de forma independente e idêntica; assim, temos:

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = \alpha \prod_{j=1}^{M} e^{-(z_j - \hat{z}_j)/2\sigma^2} .$$

A Figura 25.8(b) mostra um exemplo de varredura de intervalo de quatro feixes e duas poses

possíveis de robôs, uma das quais tem probabilidade razoável de ter produzido a varredura observada, enquanto a outra não tem.

Comparando o modelo de varredura de intervalo com o modelo de marco, vemos que o modelo de varredura de intervalo apresenta a vantagem de não haver necessidade de *identificar* um marco antes de a varredura de intervalo poder ser interpretada; na realidade, na Figura 25.8(b), o robô fica diante de uma parede sem características. Por outro lado, se *houver* um marco visível e identificável, ele poderá fornecer a localização imediata.

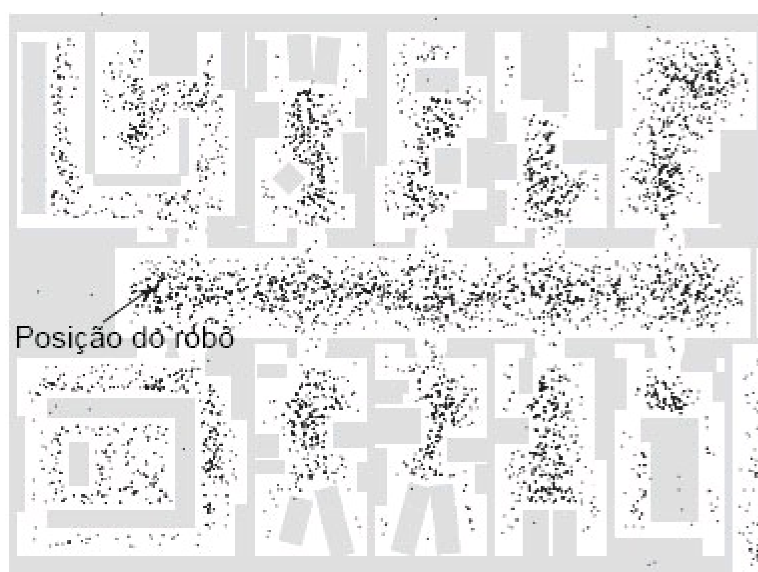
O Capítulo 15 descreveu o filtro de Kalman, que representa o estado de crença como um único gaussiano multivariado, e o filtro de partículas, que representa o estado de crença por uma coleção de partículas que correspondem a estados. A maioria dos algoritmos de localização modernos utiliza uma ou duas representações da crença do robô $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$.

A localização com o uso de filtragem de partículas é chamada **localização de Monte Carlo**, ou LMC. O algoritmo LMC é essencialmente idêntico ao algoritmo de filtragem de partículas da Figura 15.17. Tudo o que precisamos fazer é fornecer o modelo de movimento e o modelo de sensores apropriados. A Figura 25.9 mostra uma versão que utiliza o modelo de varredura de intervalo. A operação do algoritmo é ilustrada na Figura 25.10 à medida que o robô descobre onde está, dentro de um edifício comercial. Na primeira imagem, as partículas estão uniformemente distribuídas com base na probabilidade anterior, indicando incerteza global sobre a posição do robô. Na segunda imagem, o primeiro conjunto de medições chega, e as partículas formam agrupamentos nas áreas de alta crença posterior. Na terceira, estão disponíveis medições suficientes para empurrar todas as partículas para uma única posição.

função LOCALIZAÇÃO-MONTE-CARLO ($a, z, N, P(X' \mid X, v, \omega), P(z \mid z^*), m$) **retorna** um conjunto de amostras para o próximo período de tempo
  entradas: a, velocidades do robô, v e ω
    z, intervalo de varredura $z_1, \ldots, z_M$
    $P(X' \mid X, v, \omega)$, modelo de movimento
    $P(z \mid z^*)$, modelo de ruído do sensor de alcance
    m, 2D mapa do ambiente
  persistente: S, um vetor de amostras de tamanho N
  variáveis locais: W, um vetor de pesos de tamanho N
    S', um vetor temporário de partículas de tamanho N
    W', um vetor de pesos de tamanho N
  se S está vazio, **então** /* fase de inicialização */
    para $i = 1$ até N **faça**
      $S[i] \leftarrow$ amostra de $P(X_0)$
    para $i = 1$ até N **faça** /* ciclo de atualização */
      $S'[i] \leftarrow$ amostra de $P(X' \mid X = S[i], v, \omega)$
      $W'[i] \leftarrow 1$
      para $j = 1$ até M **faça**

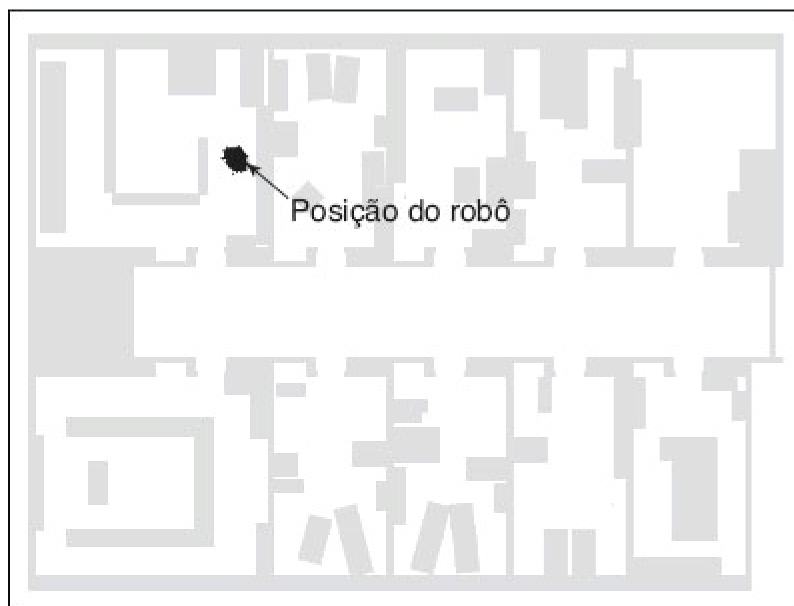
```
            z* RAYCAST (j, X = S′[i], m)
            W′ [i] ← W′ [i] · P (z_j | z*)
    S ← AMOSTRA-PONDERADA-COM-SUBSTITUIÇÃO (N, S′, W′)
retorno S
```

Figura 25.9 Algoritmo de localização de MonteCarlo que utiliza um modelo de sensores de varredura de intervalo com ruído independente.





(b)



(c)

Figura 25.10 Localização de Monte Carlo, um algoritmo de filtro de partículas para localização de

um robô móvel. (a) Incerteza inicial, global. (b) Incerteza aproximadamente bimodal após navegar no corredor (simétrico). (c) Incerteza unimodal após entrada em sala específica.

O filtro de Kalman é o outro modo importante de realizar a localização. Um filtro de Kalman representa a distribuição posterior $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ por um gaussiano. A média desse gaussiano será denotada por μ_t e sua covariância será $\sum_t$. O principal problema das convicções gaussianas é que elas só são fechadas sob modelos de movimento linear f e modelos de medição linear h. No caso de f ou h não linear, o resultado da atualização de um filtro em geral não será gaussiano. Desse modo, os algoritmos de localização que usam o filtro de Kalman **linearizam** os modelos de movimento e de sensores. A linearização é uma aproximação local de uma função não linear por uma função linear. A Figura 25.11 ilustra o conceito de linearização para um modelo (unidimensional) de movimento de robô. À esquerda, ela representa um modelo de movimento não linear $f(\mathbf{x}_t, a_t)$ (o controle a_t é omitido desse grafo, pois ele não desempenha nenhum papel na linearização). À direita, essa função é aproximada por uma função linear $\tilde{f}(\mathbf{x}_t, a_t)$. Essa função linear é tangente a f no ponto μ_t, a média de nossa estimativa de estado no tempo t. Tal linearização é chamada **expansão de Taylor** (de primeiro grau). Um filtro de Kalman que lineariza f e h via expansão de Taylor é chamado **filtro de Kalman estendido** (ou FKE). A Figura 25.12 mostra uma sequência de estimativas de um robô que executa um algoritmo de localização de filtro de Kalman estendido. À medida que o robô se move, a incerteza em sua estimativa de posição aumenta, como mostram as elipses de erro. Seu erro diminui conforme ele detecta o intervalo e o porte até um marco com posição conhecida. Por fim, o erro aumenta novamente, conforme o robô perde de vista o marco. Os algoritmos de FKE funcionam bem se os marcos são identificados com facilidade. Caso contrário, a distribuição posterior pode ser multimodal, como na Figura 25.10(b). O problema de precisar conhecer a identidade de marcos é uma instância do problema de **associação de dados** descrito no final na Figura 15.6.

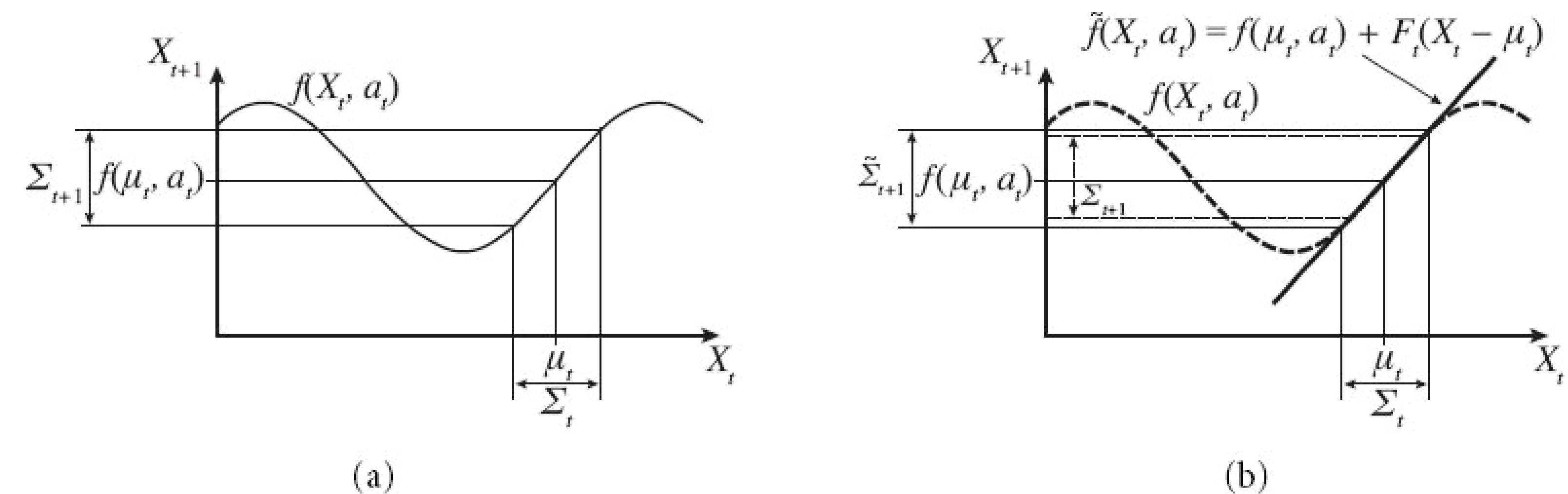


Figura 25.11 Ilustração unidimensional de um modelo de movimento linearizado: (a) a função f, a projeção de uma média μ_t. e um intervalo de covariância (baseado em $\sum_t$) para o tempo $t + 1$. (b) A versão linearizada é a tangente de f em μ_t. A projeção da média μ_t. está correta. No entanto, a covariância projetada $\sum_{t+1}$ difere de $\sum_{t+1}$.

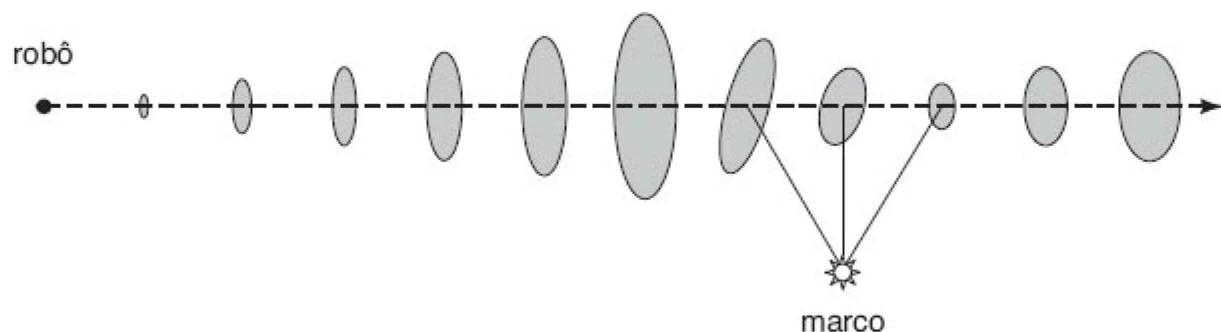


Figura 25.12 Exemplo de localização usando o filtro de Kalman estendido. O robô se move em linha reta. À medida que ele progride, sua incerteza aumenta gradualmente, conforme ilustram as elipses de erro. Quando ele observa um marco com posição conhecida, a incerteza é reduzida.

Em algumas situações, não há nenhum mapa do ambiente disponível. Então, o robô terá de adquirir um mapa. Isso é como o problema da galinha e do ovo: o robô em navegação terá de determinar a sua localização em relação a um mapa que não conhece bem, ao mesmo tempo construir esse mapa enquanto não sabe bem a sua localização real. Esse problema é importante para muitas aplicações de robô e tem sido estudado extensivamente sob o nome de **localização e mapeamento simultâneos**, abreviado como LEMS.

Os problemas de LEMS são resolvidos utilizando diversas técnicas probabilísticas, incluindo o filtro de Kalman estendido discutido antes. Utilizar o FKE é simples: basta aumentar o vetor de estado para incluir a localização dos marcos no ambiente. Felizmente, o FKE atualiza as escalas de forma quadrática; assim, para mapas pequenos (por exemplo, algumas centenas de marcos), o cálculo é bastante viável. Mapas mais ricos são geralmente obtidos utilizando métodos de relaxamento de grafo, semelhante às técnicas de inferência de rede bayesiana discutidas no Capítulo 14. A expectativa-maximização também é utilizada para LEMS.

25.3.2 Outros tipos de percepção

Nem tudo o que existe no estudo da percepção dos robôs se refere à localização e ao mapeamento. Os robôs também percebem temperatura, odores, sinais acústicos, e assim por diante. Muitas dessas quantidades podem ser estimadas utilizando variantes das redes de Bayes dinâmicas.

Tudo o que é necessário para tais avaliadores são distribuições de probabilidade condicional que caracterizem a evolução de variáveis de estados com o passar do tempo, além de outras distribuições que descrevam a relação entre as medições e as variáveis de estados.

Também é possível programar um robô como um agente reativo, sem raciocinar explicitamente sobre as distribuições de probabilidade sobre os estados. Essa abordagem será considerada na Seção 25.6.3.

A tendência em robótica é claramente orientada para representações com semântica bem definida.

As técnicas probabilísticas superam outras abordagens em muitos problemas perceptivos difíceis, como localização e mapeamento. No entanto, as técnicas estatísticas às vezes são muito incômodas, e soluções mais simples podem ser igualmente efetivas na prática. Para ajudar a decidir que abordagem adotar, a experiência prática no trabalho com robôs físicos reais sempre é o melhor

professor.

25.3.3 Aprendizagem de máquina na percepção do robô

A aprendizagem de máquina tem um papel importante na percepção do robô, particularmente no caso em que a melhor representação interna não é conhecida. Uma abordagem comum é mapear sensores de fluxo de dimensão superior em espaços de dimensão inferior utilizando métodos de aprendizagem de máquina não supervisionada (veja o Capítulo 18). Tal abordagem é chamada de **incorporação de dimensão inferior**. A aprendizagem de máquina torna possível aprender modelos de sensores e de movimento a partir de dados e, simultaneamente, descobrir as representações internas adequadas.

Outra técnica de aprendizagem de máquina permite que os robôs adaptem-se continuamente às mudanças amplas de medições do sensor. Imagine-se andando de um espaço iluminado pelo Sol para um quarto escuro iluminado por neon. É certo que os objetos são mais escuros lá dentro. Mas a mudança da fonte de luz também afeta todas as cores: a luz de neon tem um componente mais forte de luz verde do que a luz solar. No entanto, de alguma forma parece que não percebemos a mudança. Se andarmos junto com pessoas em uma sala iluminada por neon, não pense que os rostos vão se tornar verdes repentinamente. Nossa percepção se adapta rapidamente às novas condições de iluminação e nosso cérebro ignora as diferenças.

As técnicas adaptativas de percepção permitem aos robôs se ajustarem a essas mudanças. Mostramos um exemplo na Figura 25.13, tomado a partir do domínio de condução autônoma. Aqui um veículo terrestre não tripulado adapta seu classificador ao conceito de “superfície trafegável”. Como isso funciona? O robô utiliza um *laser* para fornecer classificação para uma pequena área bem em frente ao robô. Quando se conclui que essa área está no intervalo de varredura do *laser*, é utilizada como exemplo de treino positivo para o conceito de “superfície trafegável”. Uma mistura de técnicas gaussianas semelhante à do algoritmo EM discutido no Capítulo 20 é, então, treinada para reconhecer a cor específica e os coeficientes de textura da pequena amostra do trecho. As imagens na Figura 25.13 são o resultado da aplicação do classificador para a imagem completa.

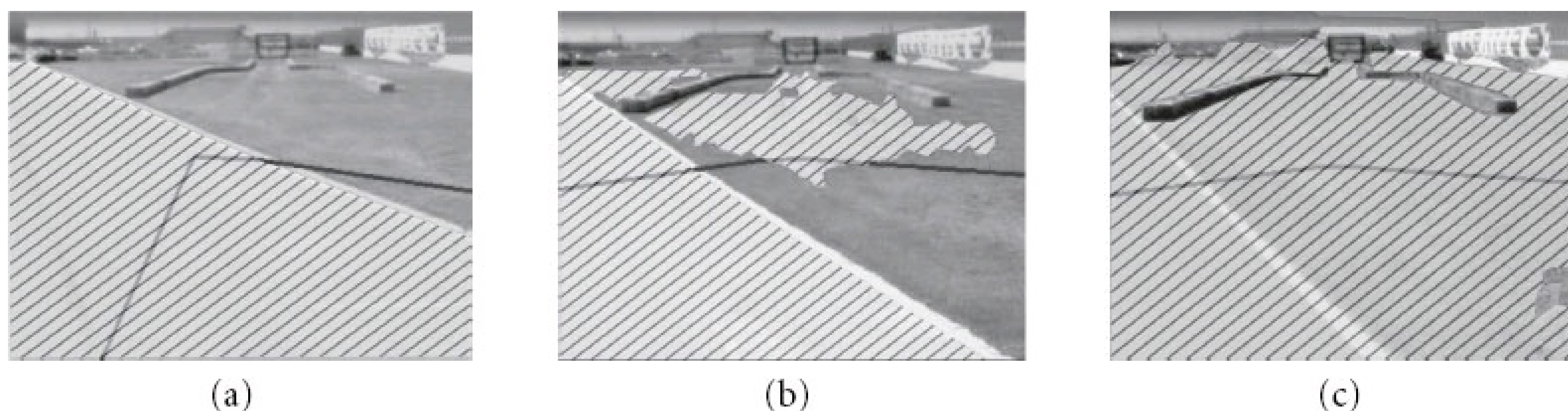


Figura 25.13 Sequência de resultados do classificador de “superfície trafegável” utilizando a visão adaptativa. Em (a) apenas a estrada é classificada como trafegável (área listrada). A linha escura em forma de V mostra para onde o veículo está dirigindo. Em (b) o veículo é comandado para dirigir fora da estrada, em uma superfície gramada, e o classificador está começando a classificar algumas

das gramíneas como trafegáveis. Em (c) o veículo atualizou seu modelo de superfície trafegável para corresponder tanto a grama como a estrada.

Métodos **autossupervisionados** são os que fazem com que os robôs recolham os seus próprios dados de treinamento (com rótulos!). Nesse exemplo, o robô utiliza aprendizagem de máquina para nivelar um sensor de curto alcance, que funciona bem para a classificação do terreno, em um sensor que pode ver muito mais longe. Isso permite que o robô se mova mais rápido, diminuindo apenas quando o modelo do sensor informa que há uma mudança no terreno que precisa ser examinada com mais cuidado pelos sensores de alcance curto.

25.4 PLANEJAMENTO DO MOVIMENTO

Todas as deliberações de um robô reduzem-se a decidir como mover efetuadores. O problema de **movimento ponto a ponto** consiste em colocar o robô ou seu efetuador final em uma posição de destino designada.

Um desafio maior é o problema de **movimento compatível**, em que um robô se movimenta enquanto está em contato físico com um obstáculo. Um exemplo de movimento compatível é um manipulador de robô capaz de girar uma lâmpada incandescente em um bocal ou um robô que empurra uma caixa sobre o tampo de uma mesa.

Começamos pela definição de uma representação apropriada, na qual os problemas de planejamento de movimento possam ser descritos e resolvidos. Concluímos que o **espaço de configuração** — o espaço de estados do robô definidos pela posição, pela orientação e pelos ângulos de articulação — é um lugar melhor para se trabalhar que o espaço tridimensional original. O problema de **planejamento de caminho** consiste em encontrar um caminho de uma configuração até outra no espaço de configuração. Já encontramos várias versões do problema de planejamento de caminho ao longo deste livro; em robótica, a principal característica do planejamento de caminho é a de que ele envolve espaços *contínuos*. Há duas abordagens principais: **de composição de células** e **esqueletização**. Cada uma delas reduz o problema de planejamento de caminho contínuo a um problema de busca em grafo discreto. Nesta seção, supomos que o movimento é determinístico e que a localização do robô é exata. As seções subsequentes relaxarão essas suposições.

25.4.1 Espaço de configuração

O primeiro passo em direção a uma solução para o problema de movimento de robôs é criar uma representação apropriada do problema. Começaremos com uma representação simples para um problema simples. Considere o braço robô mostrado na Figura 25.14(a). Ele tem duas articulações que se movem independentemente. O movimento das articulações altera as coordenadas (x, y) do cotovelo e da garra (o braço não pode se mover na direção z). Isso sugere que a configuração do robô pode ser descrita por uma coordenada quadridimensional: (x_e, y_e) para a posição do cotovelo em relação ao ambiente e (x_g, y_g) para a posição da garra. É claro que essas quatro coordenadas

caracterizam o estado completo do robô. Elas constituem o que se conhece como representação do **espaço de trabalho**, tendo em vista que as coordenadas do robô são especificadas no mesmo sistema de coordenadas que os objetos que ele busca manipular (ou evitar). As representações do espaço de trabalho são adequadas para verificação de colisões, em especial se os robôs e todos os objetos forem representados por modelos poligonais simples.

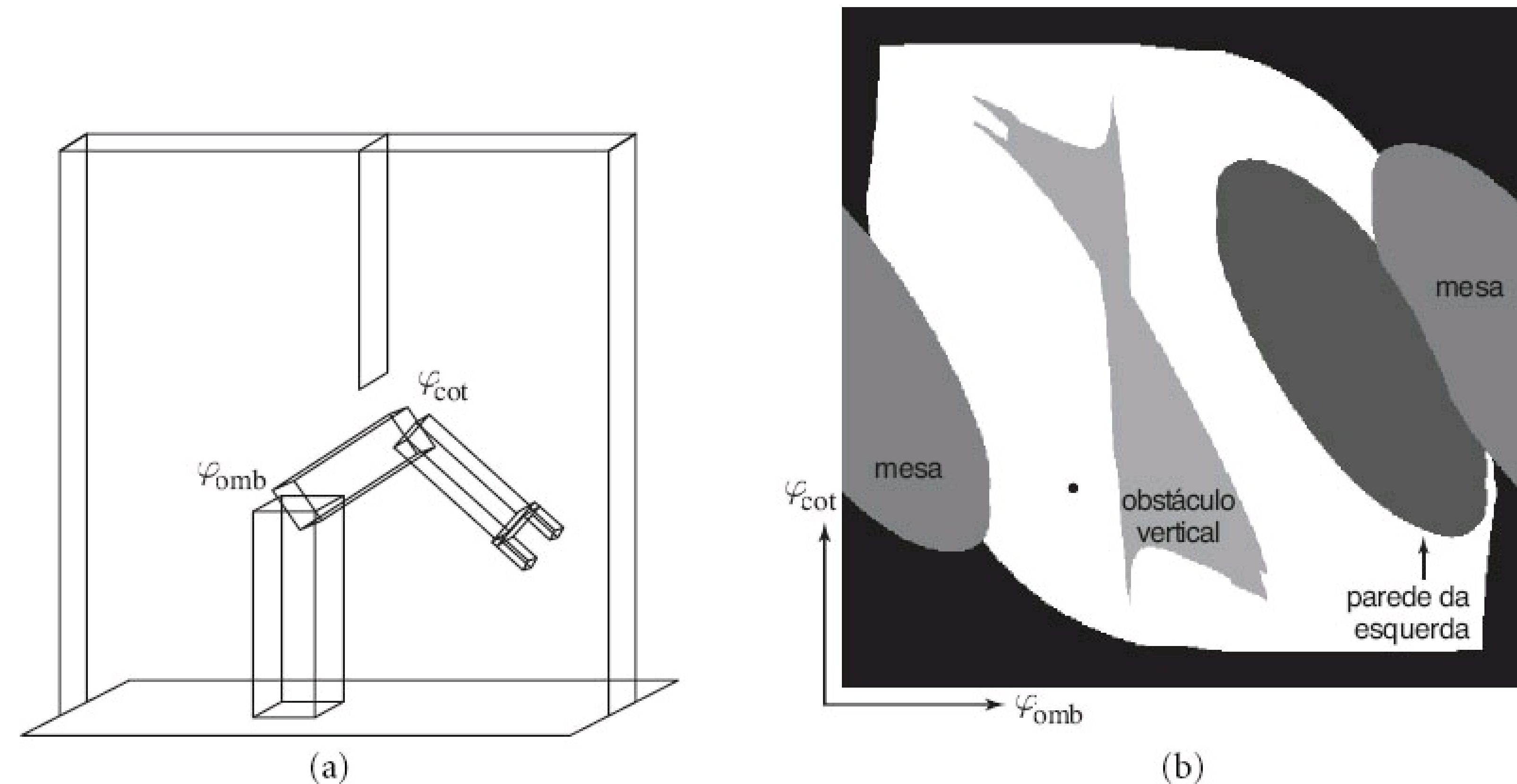


Figura 25.14 (a) Representação do espaço de trabalho de um braço robô com 2 GDLs. O espaço de trabalho é uma caixa com um obstáculo plano pendente do teto. (b) Espaço de configuração do mesmo robô. Somente as regiões brancas no espaço são configurações livres de colisões. O ponto nesse diagrama corresponde à configuração do robô mostrado à esquerda.

O problema com a representação do espaço de trabalho é que nem todas as coordenadas do espaço de trabalho são de fato acessíveis, mesmo na ausência de obstáculos. Isso ocorre devido a **restrições de vinculação** sobre o espaço de coordenadas acessíveis no espaço de trabalho. Por exemplo, a posição do cotovelo (x_e, y_e) e a posição da garra (x_g, y_g) estão sempre afastadas entre si por uma distância fixa porque eles estão unidos por um antebraço rígido. Um planejador de movimento de robô definido sobre coordenadas do espaço de trabalho enfrenta o desafio de gerar caminhos que aceitem essas restrições. Isso é particularmente complicado, porque o espaço de estados é contínuo e as restrições são não lineares.

Concluímos que é mais fácil planejar com uma representação do **espaço de configuração**. Em vez de representar o estado do robô pelas coordenadas cartesianas de seus elementos, representamos o estado por uma configuração de articulações do robô. Nosso exemplo de robô possui duas articulações. Consequentemente, podemos representar seu estado com os dois ângulos j_s e j_e referentes às articulações do ombro e do cotovelo, respectivamente. Na ausência de quaisquer obstáculos, um robô pode assumir livremente qualquer valor no espaço de configuração. Em

particular, ao planejar um caminho, poderíamos simplesmente conectar a configuração atual e a configuração de destino por uma linha reta. Seguindo esse caminho, o robô mudaria suas articulações a uma velocidade constante, até alcançar uma posição de destino.

Infelizmente, os espaços de configuração têm seus próprios problemas. A tarefa de um robô normalmente é expressa em coordenadas do espaço de trabalho, não em coordenadas do espaço de configuração. Isso nos traz a questão de como mapear essas coordenadas do espaço de trabalho no espaço de configuração. Transformar coordenadas do espaço de configuração em coordenadas do espaço de trabalho é simples: envolve uma série de transformações de coordenadas bastante óbvias. Essas transformações são lineares para articulações prismáticas, e trigonométricas para articulações de revolução. Essa cadeia de transformações de coordenadas é conhecida como **cinemática**.

O problema inverso de calcular a configuração do robô cuja posição do efetuador é especificada em coordenadas do espaço de trabalho é conhecido como **cinemática inversa**. O cálculo da cinemática inversa em geral é difícil, especialmente para robôs com muitos GDLs. Em particular, a solução raramente é única. A Figura 25.14(a) mostra uma das duas configurações possíveis que colocam a garra no mesmo local (a outra configuração teria o cotovelo abaixo do ombro).

Normalmente, esse braço robô tem entre zero e duas soluções cinemáticas inversas para qualquer conjunto de coordenadas do espaço de trabalho. A maioria dos robôs industriais tem um número infinito de soluções. Para ver como isso é possível, simplesmente imagine que adicionamos uma terceira articulação de revolução ao robô do nosso exemplo, cujo eixo rotacional é paralelo aos eixos da articulação existente. Em tal caso, podemos manter fixa a posição (mas não a orientação!) da garra e, ainda assim, girar livremente suas articulações internas, para a maioria das configurações do robô. Com mais algumas articulações (quantas?) podemos alcançar o mesmo efeito enquanto a orientação também é mantida constante. Já vimos um exemplo desse fato no “experimento” de pousar a mão sobre a escrivaninha e mover o cotovelo. A restrição cinemática da posição da mão é insuficiente para determinar a configuração do cotovelo. Em outras palavras, a cinemática inversa do conjunto ombro-braço tem um número infinito de soluções.

O segundo problema com representações do espaço de configuração surge em consequência dos obstáculos que podem existir no espaço de trabalho do robô. Nosso exemplo na Figura 25.14(a) mostra vários desses obstáculos, inclusive um obstáculo pendente livremente do teto que se projeta para o centro do espaço de trabalho do robô. No espaço de trabalho, tais obstáculos assumem formas geométricas simples — em especial, na maioria dos livros didáticos sobre robótica, que tendem a se concentrar em obstáculos poligonais. Porém, qual é sua aparência no espaço de configuração?

A Figura 25.14(b) mostra o espaço de configuração para nosso exemplo de robô, sob a configuração específica de obstáculo mostrada na Figura 25.14(a). O espaço de configuração pode ser decomposto em dois subespaços: o espaço de todas as configurações que um robô pode atingir, comumente chamado **espaço livre**, e o espaço de configurações inacessíveis, chamado **espaço ocupado**. A área branca na Figura 25.14(b) corresponde ao espaço livre. Todas as outras regiões correspondem ao espaço ocupado. O sombreamento diferente do espaço ocupado corresponde aos diferentes objetos no espaço de trabalho do robô; a região preta que cerca todo o espaço livre corresponde a configurações em que o robô colide consigo mesmo. É fácil ver que os valores extremos dos ângulos do ombro ou do cotovelo causam tal violação. As regiões de forma oval em ambos os lados do robô correspondem à mesa sobre a qual o robô está montado. De modo

semelhante, a terceira região oval corresponde à parede da esquerda. Finalmente, o objeto mais interessante no espaço de configuração é o obstáculo vertical simples que obstrui o espaço de trabalho do robô. Esse objeto tem uma forma interessante: ele éaltamente não linear e, em certos lugares, até mesmo côncavo. Com um pouco de imaginação, o leitor reconhecerá a forma da garra no canto superior esquerdo. Encorajamos o leitor a fazer uma pausa por um momento e estudar esse importante diagrama. A forma desse obstáculo não é de modo algum óbvia! O ponto dentro da Figura 25.14(b) marca a configuração do robô, como mostra a Figura 25.14(a). A Figura 25.15 representa três configurações adicionais, ambas no espaço de trabalho e no espaço de configuração. Na configuração “conf-1”, a garra envolve o obstáculo vertical.

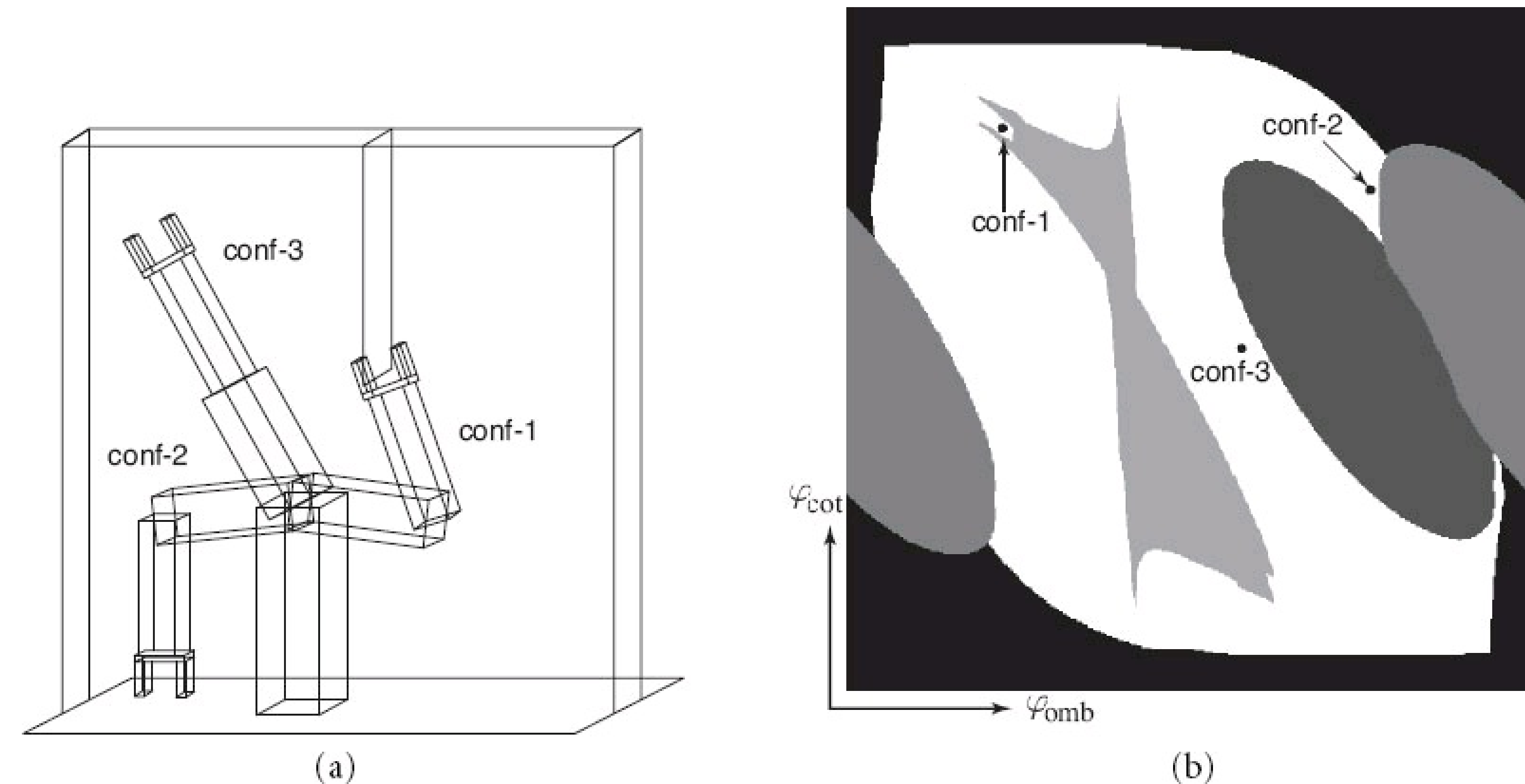


Figura 25.15 Três configurações de um robô mostradas no espaço de trabalho e no espaço de configuração.

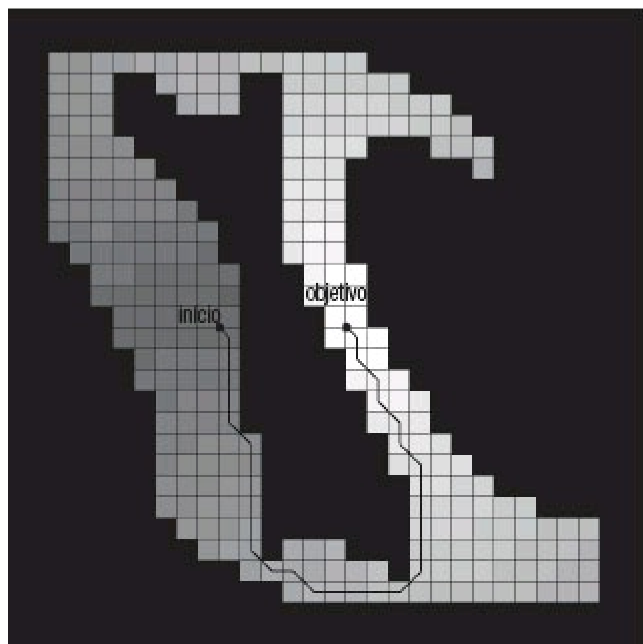
Em geral, mesmo que o espaço de trabalho do robô esteja representado por polígonos planos, a forma do espaço livre pode ser muito complicada. Portanto, na prática costuma-se *sondar* um espaço de configuração, em vez de construí-lo explicitamente. Um planejador pode gerar uma configuração e depois testá-la para ver se ela está no espaço livre, aplicando a cinemática do robô e, em seguida, verificando se há colisões em coordenadas do espaço de trabalho.

25.4.2 Métodos de decomposição em células

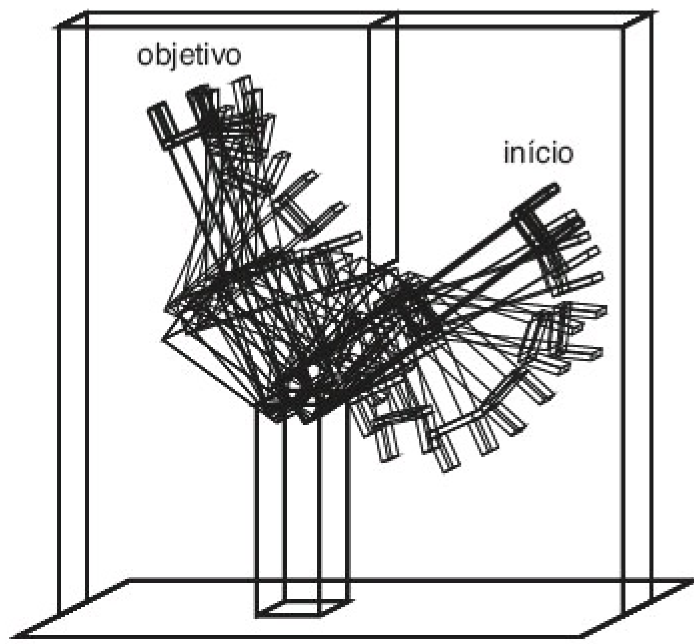
A primeira abordagem para planejamento de caminho utiliza **decomposição em células**, isto é, decompõe-se o espaço livre em um número finito de regiões contíguas, chamadas células. Essas regiões têm a importante propriedade segundo a qual o problema de planejamento de caminho dentro de uma única região pode ser resolvido por meios simples (por exemplo, pelo movimento ao longo

de uma reta linha). Então, o problema de planejamento de caminho se torna um problema de busca em grafo discreto, muito semelhante aos problemas de busca introduzidos no Capítulo 3.

A decomposição em células mais simples consiste em uma grade regularmente espaçada. A Figura 25.16(a) mostra uma decomposição de grade quadrada do espaço e um caminho de solução que é ótimo para esse tamanho de grade. Também usamos o sombreamento em escala de cinza para indicar o *valor* de cada célula de grade do espaço livre, isto é, o custo do caminho mais curto a partir daquela célula até o objetivo (esses valores podem ser calculados por uma forma determinística do algoritmo ITERAÇÃO-DE-VALOR, dado na Figura 17.4). A Figura 25.16(b) mostra a trajetória do espaço de trabalho correspondente ao braço.



(a)



(b)

Figura 25.16 (a) Função de valor e caminho encontrados para uma aproximação de célula de grade discreta do espaço de configuração. (b) O mesmo caminho visualizado em coordenadas do espaço de trabalho. Note como o robô curva seu cotovelo para evitar uma colisão com o obstáculo vertical.

Tal decomposição tem a vantagem de ser extremamente simples de implementar, mas também sofre três limitações. Primeiro, ela só é funcional para os espaços de configuração de baixo número de dimensões, pois o número de células da grade aumenta exponencialmente com d, o número de dimensões. Parece familiar? É a maldição da dimensionalidade. Em segundo lugar, existe o problema do que fazer com células "misturadas", ou seja, células que não estão inteiramente no espaço livre nem inteiramente no espaço ocupado. Um caminho de solução que inclui tal célula não pode ser uma solução real porque pode não haver nenhum modo de cruzar a célula na direção desejada em linha reta. Isso tornaria o planejador de caminho *não consistente*. Por outro lado, se insistirmos em que só podem ser usadas células completamente livres, o planejador será *incompleto* porque poderá surgir uma situação em que os únicos caminhos para o objetivo passem por células misturadas — em especial se o tamanho da célula for comparável ao das passagens e folgas no espaço. Terceiro, qualquer caminho através de um espaço de estados discretizado não será suave. Em geral, é difícil garantir que exista uma solução suave perto do caminho discreto. Assim, um robô não pode ser capaz de executar a solução encontrada por essa decomposição.

Os métodos de decomposição de células podem ser melhorados em vários aspectos, para aliviar um pouco esses problemas. O primeiro é permitir a *subdivisão adicional* das células misturadas — talvez usando células com metade do tamanho original. Essa subdivisão pode prosseguir recursivamente até ser encontrado um caminho que resida completamente no interior de células livres. (É claro que o método só funcionará se houver um modo de descobrir se determinada célula é uma célula misturada, o que só será fácil se os limites do espaço de configuração tiverem descrições matemáticas relativamente simples.) Esse método é completo desde que exista um limite sobre a menor passagem que uma solução deve percorrer. Embora concentre a maior parte do esforço computacional nas áreas complicadas dentro do espaço de configuração, o método ainda apresenta falhas quando tem de ampliar sua escala para problemas com números de dimensões mais altos porque cada divisão recursiva de uma célula cria 2^d células menores. Um segundo caminho para obter um algoritmo completo é insistir em uma **decomposição exata em células** do espaço livre. Esse método tem de permitir que as células tenham forma irregular nos lugares em que são adjacentes aos limites do espaço livre, mas as formas ainda precisam ser "simples", no sentido de que deve ser fácil calcular um percurso que passe por qualquer célula livre.

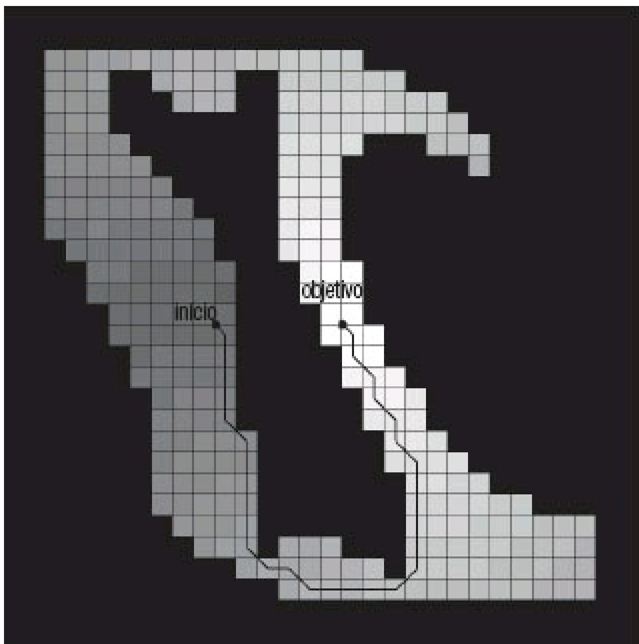
Essa técnica exige algumas ideias geométricas bastante avançadas; por esse motivo, não vamos mais examiná-la aqui.

Examinando o caminho de solução mostrado na Figura 25.16(a), podemos observar dificuldades adicionais que terão de ser resolvidas. Arbitrariamente, o caminho contém cantos afiados; um robô móvel em qualquer velocidade finita não pode executar esse caminho. Esse problema é resolvido através do armazenamento de certos valores contínuos para cada célula da grade. Considere um algoritmo que armazena, para cada célula da grade, o estado exato, contínuo, que foi atingido com a célula que foi expandida primeiro na pesquisa. Suponha ainda que, quando propagar informação para as células da grade nas proximidades, utilizaremos esse estado contínuo como base e aplicaremos o modelo de movimento contínuo do robô para saltar para as células vizinhas. Ao fazer isso, podemos agora garantir que a trajetória resultante é suave e pode realmente ser executada pelo robô. **A* Híbrido** é um algoritmo que implementa isso.

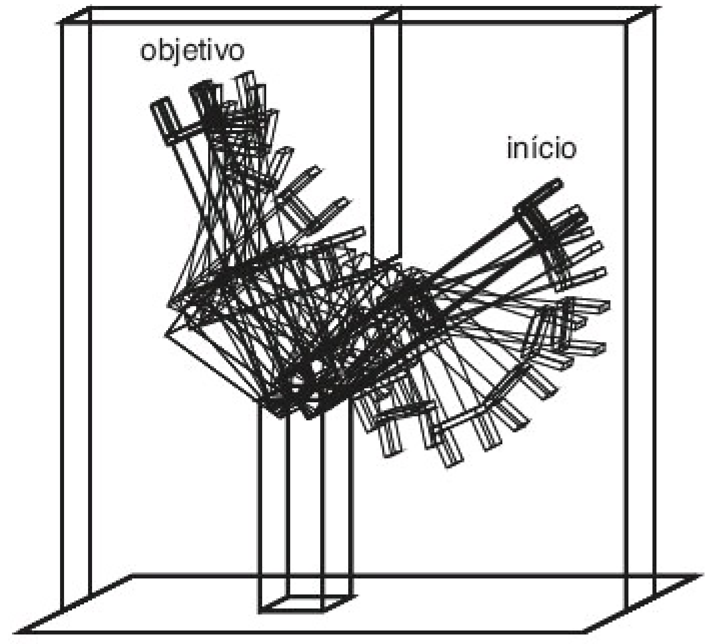
25.4.3 Funções de custo modificadas

Observe que, na Figura 25.16, o caminho passa muito perto do obstáculo. Qualquer um que tenha conduzido um carro sabe que um espaço de estacionamento com um milímetro de distância de cada lado não é realmente um espaço de estacionamento; pela mesma razão, preferimos caminhos de solução que sejam robustos com respeito a erros pequenos de movimentos.

Esse problema pode ser resolvido através da introdução de um **campo potencial**. Um campo potencial é uma função definida sobre o espaço de estados, cujo valor cresce com a distância para o obstáculo mais próximo. A Figura 25.17(a) mostra tal campo potencial — quanto mais escuro um estado de configuração, mais próximo está de um obstáculo.



(a)



(b)

Figura 25.17 (a) Um campo potencial de repulsão empurra o robô para longe dos obstáculos. (b) O caminho encontrado minimizando-se simultaneamente o comprimento do caminho e o potencial.

O campo potencial pode ser usado como um termo de custo adicional no cálculo do caminho mais curto. Isso induz a uma troca interessante. Por um lado, o robô procura minimizar o comprimento do caminho até o objetivo. Por outro lado, tenta ficar longe de obstáculos em virtude de minimizar a função potencial. Com o peso adequado equilibrando os dois objetivos, um caminho resultante pode parecer com o mostrado na Figura 25.17(b). Essa figura também apresenta a função de valor derivada da função custo combinada, calculada novamente por iteração de valor. Certamente o caminho resultante será mais longo, mas também mais seguro.

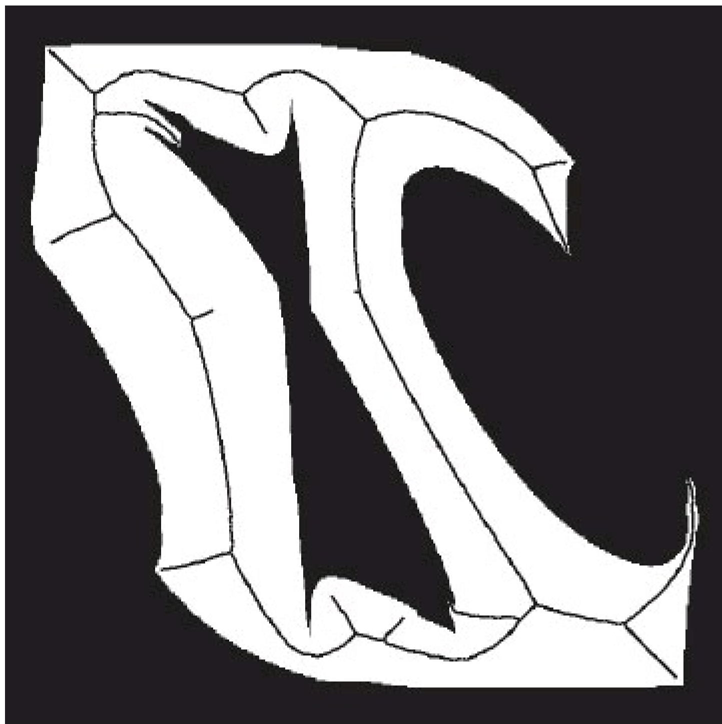
Existem muitas outras maneiras de modificar a função de custo. Por exemplo, pode ser desejável *alisar* os parâmetros de controle ao longo do tempo. Por exemplo, ao dirigir um carro, um caminho suave é melhor do que um aos trancos e barrancos. Em geral, tais restrições de ordem superior não são fáceis de acomodar no processo de planejamento, a menos que tornemos o comando de direção mais recente uma parte do estado. No entanto, muitas vezes é fácil suavizar a trajetória resultante após o planejamento, utilizando métodos de gradiente conjugado. A suavização de pós-planejamento é essencial em muitas aplicações do mundo real.

25.4.4 Métodos de esqueletização

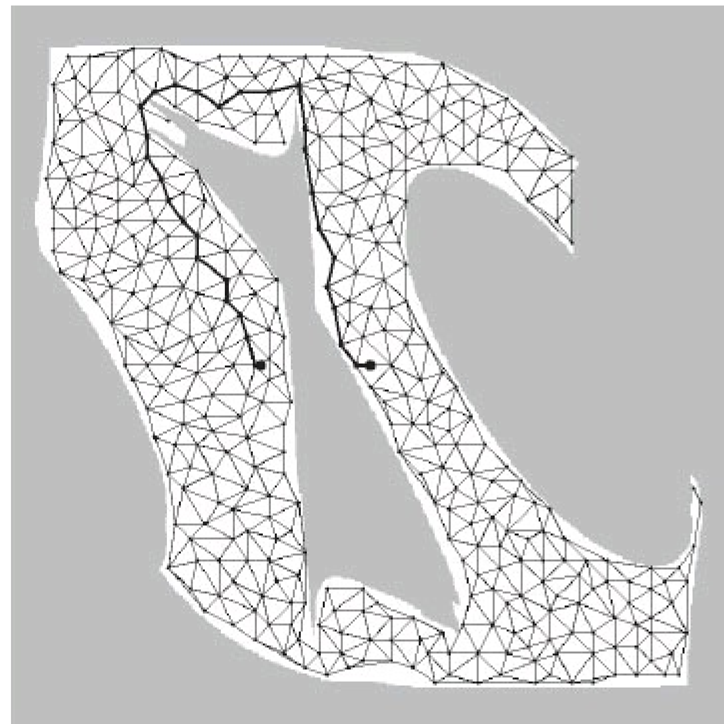
A segunda família importante de algoritmos de planejamento de caminho se baseia na ideia de **esqueletização**. Esses algoritmos reduzem o espaço livre do robô a uma representação unidimensional, para a qual o problema de planejamento é mais fácil. Essa representação com número mais baixo de dimensões é denominada **esqueleto** do espaço de configuração.

A Figura 25.18 mostra um exemplo de esqueletização: ela representa um **diagrama de Voronoi** do espaço livre — o conjunto de todos os pontos equidistantes de dois ou mais obstáculos. Para fazer o planejamento de caminho com um diagrama de Voronoi, primeiro o robô muda sua configuração atual

para um ponto no diagrama de Voronoi. É fácil mostrar que isso sempre pode ser alcançado por meio de um movimento em linha reta no espaço de configuração. Em segundo lugar, o robô segue o diagrama de Voronoi até alcançar o ponto mais próximo à configuração de destino. Finalmente, o robô deixa o diagrama de Voronoi e se move até o destino. Mais uma vez, essa última etapa envolve o movimento em linha reta no espaço de configuração.



(a)



(b)

Figura 25.18 (a) O diagrama de Voronoi é o conjunto de pontos equidistantes de dois ou mais obstáculos no espaço de configuração. (b) Roteiro probabilístico, composto de 400 pontos escolhidos ao acaso no espaço livre.

Desse modo, o problema original de planejamento de caminho é reduzido à descoberta de um caminho no diagrama de Voronoi, que em geral é unidimensional (exceto em alguns casos não genéricos) e tem um número finito de pontos em que se interceptam três ou mais curvas unidimensionais. Desse modo, encontrar o caminho mais curto ao longo do diagrama de Voronoi é um problema de busca em grafos discretos do tipo descrito nos Capítulos 3 e 4. Seguir o diagrama de Voronoi talvez não nos dê o caminho mais curto, mas os caminhos resultantes tenderão a maximizar a folga. As desvantagens das técnicas de diagramas de Voronoi estão na dificuldade de sua aplicação a espaços de configuração de dimensões mais altas e ainda no fato de que eles tendem a induzir desvios desnecessariamente grandes quando o espaço de configuração é aberto. Além disso, pode ser difícil calcular o diagrama de Voronoi, especificamente no espaço de configuração, onde a forma de obstáculos pode ser complexa.

Uma alternativa para os diagramas de Voronoi é o **roteiro probabilístico**, uma abordagem de esqueletização que oferece mais rotas possíveis e, desse modo, lida melhor com os espaços abertos amplos. A Figura 25.18(b) mostra um exemplo de roteiro probabilístico. O grafo é criado gerando-se aleatoriamente grande número de configurações e descartando-se aquelas que não se enquadram no espaço livre. Então, unimos dois nós quaisquer por um arco, se for "fácil" alcançar um nó a partir do outro — por exemplo, por uma linha reta no espaço livre. O resultado de tudo isso é um grafo

aleatório no espaço livre do robô. Se adicionarmos as configurações de início e de destino do robô a esse grafo, o planejamento de caminho se reduzirá a uma busca no grafo discreto. Teoricamente, essa abordagem é incompleta porque uma escolha ruim de pontos aleatórios pode nos deixar sem quaisquer caminhos do início até o destino. É possível limitar a probabilidade de falha em termos do número de pontos gerados e certas propriedades geométricas do espaço de configuração. Também é possível orientar a geração de pontos de amostra para as áreas em que uma busca parcial sugere que poderá ser encontrado um bom caminho trabalhando-se nos dois sentidos a partir da posição inicial e da posição objetivo. Com essas melhorias, o planejamento de roteiro probabilístico tende a ajustar melhor sua escala a espaços de configuração de dimensões mais altas que a maioria das técnicas alternativas de planejamento de caminho.

25.5 PLANEJAMENTO DE MOVIMENTOS INCERTOS

Nenhum dos algoritmos de planejamento de movimento de robô descritos até agora estuda uma característica fundamental dos problemas de robótica: a *incerteza*. Em robótica, a incerteza surge a partir da observabilidade parcial do ambiente e a partir dos efeitos estocásticos (ou não modelados) das ações do robô. Os erros também podem surgir do uso de algoritmos de aproximação como filtragem de partículas, que não fornecem ao robô um estado de crença exato, mesmo que a natureza estocástica do ambiente seja perfeitamente modelada.

A maior parte dos robôs atuais utiliza algoritmos determinísticos para tomada de decisões, como os diversos algoritmos de planejamento de caminho que descrevemos até este momento. Para isso, é prática comum extrair o **estado mais provável** da distribuição de estados produzida pelo algoritmo de localização. A vantagem dessa abordagem é puramente computacional. O planejamento de caminhos pelo espaço de configuração já é um problema desafiador; seria pior se tivéssemos de trabalhar com uma distribuição de probabilidade completa sobre estados. Desse modo, ignorar a incerteza funciona bem quando a incerteza é pequena. Na verdade, quando o modelo do ambiente muda ao longo do tempo, como resultado de incorporar as medições do sensor, muitos robôs planejam caminhos on-line durante a execução do plano. Essa é a técnica do **replanejamento on-line** da Seção 11.3.3.

Infelizmente, ignorar a incerteza nem sempre funciona. Em alguns problemas, a incerteza do robô é simplesmente muito grande: como podemos usar um planejador de caminho determinístico para controlar um robô móvel que não tem nenhuma pista de onde está? Em geral, se o verdadeiro estado do robô não é aquele identificado pela regra de máxima probabilidade, o controle resultante será não ótimo. Dependendo da magnitude do erro, isso poderá levar a todos os tipos de efeitos indesejáveis, como colisões com obstáculos.

O campo da robótica adotou uma variedade de técnicas para acomodar a incerteza. Algumas são derivadas dos algoritmos dados no Capítulo 17 para tomada de decisões sob incerteza. Se o robô só se defrontar com a incerteza em sua transição de estados, mas seu estado for completamente observável, o problema será mais bem modelado como um processo de decisão de Markov, ou MDP. A solução de um MDP é uma **política** ótima, que informa ao robô o que fazer em cada estado possível. Desse modo, ele pode manipular todos os tipos de erros de movimentação enquanto uma

solução de caminho único de um planejador determinístico seria muito menos resistente. Em robótica, as políticas costumam ser chamadas **funções de navegação**. A função de valor mostrada na Figura 25.16(a) pode ser convertida em uma função de navegação desse tipo simplesmente seguindo-se o gradiente.

Como no Capítulo 17, a observabilidade parcial torna o problema muito mais difícil. O problema de controle de robô resultante é um MDP parcialmente observável, ou POMDP. Em tais situações, normalmente o robô mantém um estado de crença interna, como os que vimos na Seção 25.3. A solução para um POMDP é uma política definida sobre o estado de crença do robô. Em outras palavras, a entrada para a política é uma distribuição inteira de probabilidade. Isso permite ao robô basear sua decisão não apenas no que ele conhece, mas também naquilo que não conhece. Por exemplo, se estiver inseguro em relação a uma variável de estado crítica, ele poderá invocar racionalmente uma **ação de obtenção de informações**. Isso é impossível na estrutura de MDP, pois os MDPs pressupõem observabilidade completa. Infelizmente, técnicas que resolvem POMDPs de maneira exata são inaplicáveis à robótica — não existe nenhuma técnica conhecida para espaços contínuos. A discretização produz POMDPs que são de longe grandes para manusear. Um remédio é tornar a minimização da incerteza um controle objetivo. Por exemplo, a heurística de **navegação costeira** exige que o robô fique próximo a marcos conhecidos, a fim de diminuir a incerteza de sua pose. Outra abordagem se aplica às variantes do método de planejamento de roteiro probabilístico para a representação do espaço de crença. Tais métodos tendem a representar em escala melhor POMDPs discretos grandes.

25.5.1 Métodos robustos

A incerteza também pode ser tratada com o uso dos chamados métodos de **controles robustos**, em lugar de métodos probabilísticos. Um método robusto é aquele que pressupõe uma quantidade *limitada* de incerteza em cada aspecto de um problema, mas não atribui probabilidades a valores dentro do intervalo permitido. Uma solução robusta é aquela que funciona independentemente dos valores reais que ocorrem, desde que eles estejam dentro do intervalo pressuposto. Uma forma extrema de método robusto é a abordagem de **planejamento com conformação** apresentada no Capítulo 11 — ela produz planos que funcionam sem quaisquer informações sobre o estado.

Aqui, vamos examinar um método robusto utilizado para **planejamento de movimento fino** (ou PMF) em tarefas de montagem robótica. O planejamento de movimento fino envolve a movimentação do braço de um robô muito próximo a um objeto estático do ambiente. A principal dificuldade do planejamento de movimento fino é que os movimentos exigidos e as características relevantes do ambiente são muito pequenos. Em escalas tão pequenas, o robô é incapaz de medir ou controlar sua posição com precisão e também pode estar inseguro quanto à forma do próprio ambiente; vamos supor que essas incertezas sejam todas limitadas. As soluções para problemas de PMF em geral serão planos ou políticas condicionais que farão uso da realimentação de sensores durante a execução e que oferecem a garantia de funcionamento em todas as situações consistentes com os limites de incerteza assumidos.

Um plano de movimento fino consiste em uma série de **movimentos protegidos**. Cada movimento

protegido consiste em (1) um comando de movimento e (2) uma condição de término, que é um predicado sobre os valores de sensores do robô, e retorna verdadeiro para indicar o fim do movimento protegido. Em geral, os comandos de movimento são **movimentos compatíveis** que permitem ao robô deslizar, caso o comando de movimento possa provocar a colisão com um obstáculo. Como exemplo, a Figura 25.19 mostra um espaço de configuração bidimensional com um orifício vertical estreito. Ele poderia representar o espaço de configuração para a inserção de uma cavilha retangular em um orifício ligeiramente maior. Os comandos de movimento são velocidades constantes. As condições de término são o contato com uma superfície. Para modelar a incerteza de controle, vamos supor que, em vez de se mover na direção determinada pelo comando, o robô efetua um movimento real que reside no cone C_v sobre ele. A figura mostra o que aconteceria se nosso comando representasse uma velocidade vertical para baixo, a partir da região inicial *s*. Devido à incerteza na velocidade, o robô poderia se mover para qualquer lugar na envoltória cônica, possivelmente entrando no orifício, mas, com probabilidade maior, parando em algum ponto ao lado dele. Nesse caso, como o robô não saberia em que lado do orifício parou, não saberia que caminho seguir.

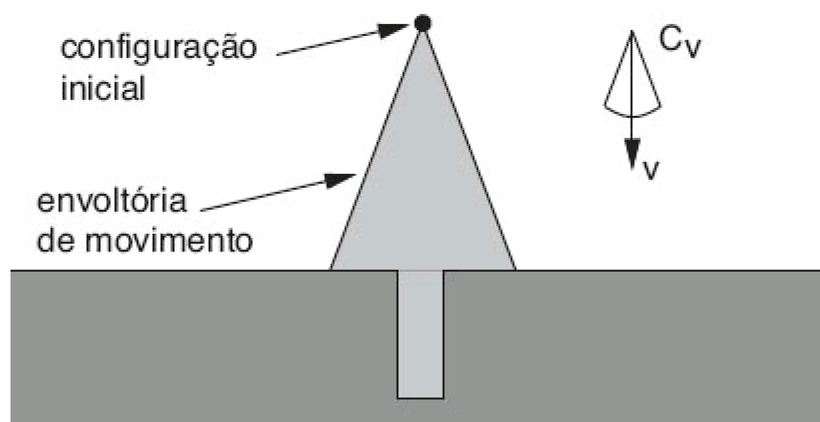


Figura 25.19 Ambiente bidimensional, cone de incerteza de velocidade e envoltória de movimentos possíveis de um robô. A velocidade pretendida é *v*, mas, com a incerteza, a velocidade real pode ter qualquer valor dentro de C_v, resultando em uma configuração final em algum lugar na envoltória de movimento; isso significa que não saberíamos se atingimos o orifício ou não.

Uma estratégia mais sensata é mostrada nas Figuras 25.20 e 25.21. Na Figura 25.20, o robô deliberadamente se move para um lado do orifício. O comando de movimento é mostrado na figura, e o teste de término é o contato com qualquer superfície. Na Figura 25.21, é executado um comando de movimento que faz o robô deslizar ao longo da superfície para dentro do orifício. Isso pressupõe que utilizamos um comando de movimento compatível. Como todas as velocidades possíveis na envoltória de movimento estão à direita, o robô deslizará para a direita sempre que estiver em contato com uma superfície horizontal. Ele deslizará para baixo da aresta vertical do lado direito do orifício quando a tocar porque todas as velocidades possíveis estarão orientadas para baixo em relação a uma superfície vertical. Ele continuará a se mover até alcançar a parte inferior do orifício porque essa é sua condição de término. Apesar da incerteza de controle, todas as trajetórias possíveis do robô terminam em contato com a parte inferior do orifício, isto é, a menos que as irregularidades da superfície façam o robô permanecer em um único lugar.

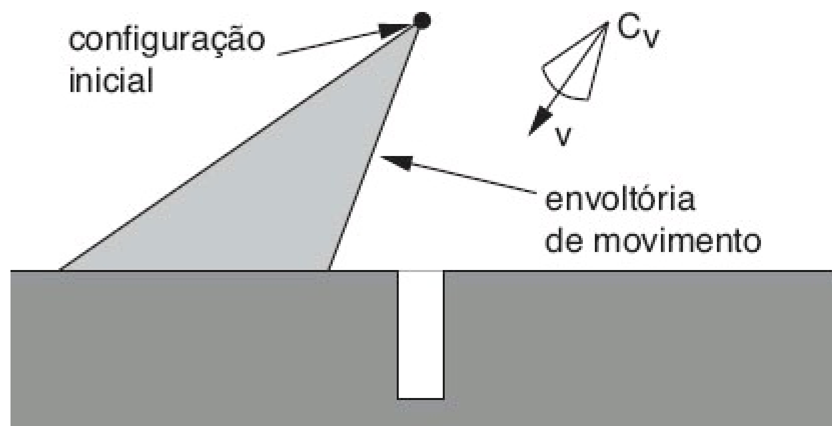


Figura 25.20 O primeiro comando de movimento e a envoltória resultante de movimentos possíveis do robô. Independentemente de qual seja o erro, sabemos que a configuração final estará à esquerda do orifício.

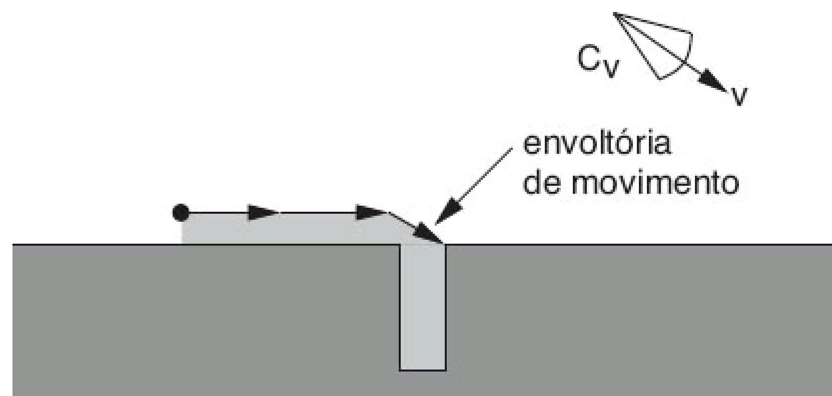


Figura 25.21 O segundo comando de movimento e a envoltória de movimentos possíveis. Mesmo com erro, eventualmente conseguiremos entrar no orifício.

Como se poderia imaginar, o problema de *construir* planos de movimento fino não é trivial; na realidade, é bem mais difícil que o planejamento com movimentos exatos. Pode-se escolher um número fixo de valores discretos para cada movimento ou usar a geometria do ambiente com o objetivo de escolher direções que resultem em um comportamento qualitativamente diferente. Um planejador de movimento fino toma como entrada a descrição do espaço de configuração, o ângulo do cone de incerteza de velocidade e uma especificação de qual detecção é possível para indicar o término (nesse caso, o contato com a superfície). Ele deve produzir um plano ou uma política condicional de várias etapas que ofereça a garantia de sucesso, se tal plano existir.

Nosso exemplo supõe que o planejador tem um modelo exato do ambiente, mas é possível permitir um erro limitado nesse modelo, como a seguir. Se o erro puder ser descrito em termos de parâmetros, esses parâmetros poderão ser adicionados como graus da liberdade ao espaço de configuração. No último exemplo, se a profundidade e a largura do orifício fossem incertas, poderíamos acrescentá-las sob a forma de dois graus de liberdade ao espaço de configuração. É impossível mover o robô nessas direções no espaço de configuração ou detectar diretamente sua posição. Porém, essas restrições podem ser incorporadas ao se descrever esse problema como um problema de PMF, especificando-se as incertezas de controle e de sensores de maneira apropriada. Isso resulta em um problema de planejamento quadridimensional complexo, mas podem ser aplicadas exatamente as mesmas técnicas de planejamento. Note que, diferentemente dos métodos de teoria da decisão do Capítulo 17, esse tipo de abordagem robusta resulta em planos projetados para o resultado no pior

caso, em vez de maximizar a qualidade esperada do plano. Os planos de pior caso só são ótimos no sentido da teoria da decisão se a falha durante a execução for muito pior que qualquer dos outros custos envolvidos na execução.

25.6 MOVIMENTO

Até agora, mencionamos como *planejar* movimentos, mas não mencionamos como se *mover*. Nossos planos — em particular, aqueles produzidos por planejadores de caminhos determinísticos — pressupõem que o robô pode simplesmente seguir qualquer caminho que o algoritmo produzir. É claro que, no mundo real, isso não acontece. Os robôs têm inércia e não podem percorrer caminhos arbitrários, exceto em velocidades arbitrariamente lentas. Na maioria dos casos, o robô chega a exercer forças, em vez de especificar posições. Esta seção descreve métodos para calcular essas forças.

25.6.1 Dinâmica e controle

A Seção 25.2 introduziu a noção de **estado dinâmico**, que estende o estado cinemático de um robô modelando as velocidades do robô. Por exemplo, além do ângulo de uma articulação do robô, o estado dinâmico também capta a taxa de mudança do ângulo e, possivelmente, até a sua aceleração momentânea. O modelo de transição para uma representação do estado dinâmico inclui o efeito de forças sob essa taxa de mudança. Tais modelos em geral são expressos por meio de **equações diferenciais**, que são equações que relacionam uma quantidade (por exemplo, um estado cinemático) à mudança da quantidade ao longo do tempo (por exemplo, a velocidade). Em princípio, poderíamos ter escolhido planejar o movimento do robô usando modelos dinâmicos, em lugar dos nossos modelos cinemáticos. Tal metodologia levaria a um desempenho superior do robô se pudéssemos gerar os planos. Porém, o estado dinâmico é mais complexo que o espaço cinemático, e a maldição da dimensionalidade geraria problemas de planejamento de movimento intratáveis para todos os robôs, exceto os mais simples. Por essa razão, os sistemas práticos de robôs frequentemente se baseiam em planejadores de caminhos cinemáticos mais simples.

Uma técnica comum para compensar as limitações de planos cinemáticos é utilizar um mecanismo separado, denominado **controlador**, para manter o robô no caminho correto. Os controladores são técnicas empregadas para gerar controles de robôs em tempo real, usando o *feedback* do ambiente, a fim de atingir um objetivo de controle. Se o objetivo for manter o robô em um caminho previamente definido, com frequência ele se chamará **controlador de referência**, e o caminho será denominado **caminho de referência**. Os controladores que otimizam uma função de custo global são conhecidos como **controladores ótimos**.

As políticas ótimas para MDPs são, na realidade, controladores ótimos. Quando o examinamos apenas superficialmente, o problema de manter um robô em um caminho pré-especificado parece ser relativamente simples. Porém, na prática, até mesmo esse problema aparentemente simples tem suas armadilhas. A Figura 25.22(a) ilustra o que pode dar errado; representa a trajetória de um robô que

tenta seguir um caminho cinemático. Sempre que acontece um desvio — seja devido ao ruído ou a restrições sobre as forças que o robô pode aplicar —, o robô exerce uma força no sentido oposto cuja magnitude é proporcional a esse desvio. Intuitivamente, isso talvez pareça plausível, tendo em vista que os desvios devem ser compensados por uma força em sentido contrário para manter o robô no caminho. Porém, como ilustra a Figura 25.22(a), nosso controlador faz o robô vibrar de maneira bastante violenta. A vibração é o resultado de uma inércia natural do braço robô: uma vez afastado de sua posição de referência, o robô se excede, o que induz um erro simétrico com sinal contrário. Tal excedente pode continuar ao longo de uma trajetória inteira, e o movimento resultante do robô fica longe de ser o desejável.

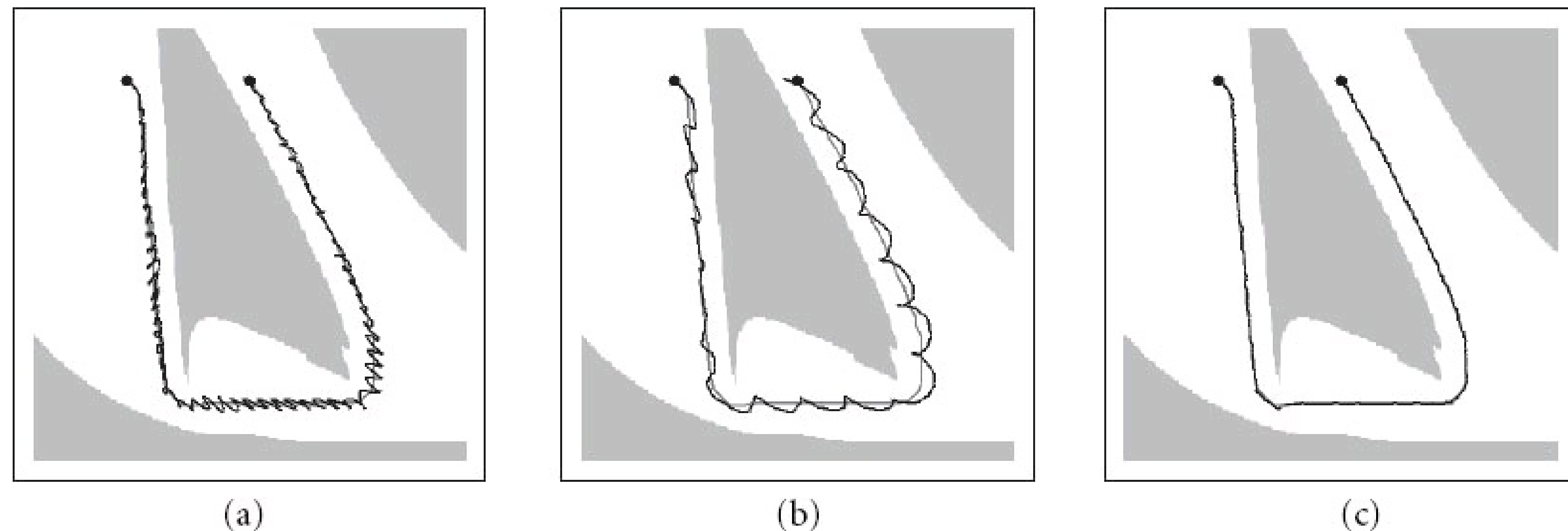


Figura 25.22 Controle de braço robô que utiliza (a) controle proporcional com fator de ganho 1,0, (b) controle proporcional com fator de ganho 0,1 e (c) controle PD com fatores de ganho 0,3 para o componente proporcional e 0,8 para o componente diferencial. Em todos os casos, o braço robô tenta seguir o caminho mostrado em cinza.

Antes de podermos definir um controle melhor vamos descrever formalmente o que deu errado. Os controladores que fornecem uma força negativa proporcional ao erro observado são conhecidos como **controladores P**. A letra "P" significa *proporcional*, indicando que o controle real é proporcional ao erro do manipulador do robô. Mais formalmente, seja $y(t)$ o caminho de referência, parametrizado pelo índice de tempo t. O controle a_t gerado por um controlador P tem a seguinte forma:

$$a_t = K_P(y(t) - x_t).$$

Aqui x_t é o estado do robô no instante t. K_P é chamado **parâmetro de ganho** do controlador e seu valor é chamado de fator de ganho; K_p regula a intensidade com que o controlador corrige desvios entre o estado real x_t e o estado desejado $y(t)$. Em nosso exemplo, $K_P = 1$. À primeira vista, poderíamos pensar que a escolha de um valor menor para K_P iria corrigir o problema. Infelizmente, não é isso que acontece. A Figura 25.22(b) mostra uma trajetória correspondente a $K_P = 0{,}1$, exibindo ainda um comportamento oscilatório. Valores mais baixos do parâmetro de ganho podem simplesmente diminuir a oscilação, mas não resolvem o problema. Na verdade, na ausência de atrito,

o controlador P é essencialmente uma mola; assim, ele vai oscilar indefinidamente em torno de uma posição de destino fixa.

Tradicionalmente, problemas desse tipo recaem no domínio da **teoria de controle**, um campo de importância crescente para os pesquisadores em IA. Décadas de pesquisa nesse campo levaram a grande número de controladores que têm qualidade superior à do controle simples explicado antes. Em particular, um controlador de referência é dito **estável** se pequenas perturbações levam a um erro limitado entre o robô e o sinal de referência. O controlador de referência é **estritamente estável** se pode retornar a seu caminho de referência ao ocorrerem tais perturbações. Sem dúvida, nosso controlador P parece ser estável, mas não estritamente estável, pois deixa de retornar para sua trajetória de referência.

O controlador mais simples que alcança estabilidade estrita em nosso domínio é conhecido como **controlador PD**. Mais uma vez, a letra "P" significa *proporcional*, e "D" significa *derivada*. Os controladores PD são descritos pela equação a seguir:

$$a_t = K_P(y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t} . \qquad (25.2)$$

Como sugere essa equação, os controladores PD estendem os controladores P com um componente diferencial, que soma ao valor de a_t um termo proporcional à primeira derivada do erro $y(t) - x_t$ ao longo do tempo.

Qual é o efeito de tal termo? Em geral, um termo derivado amortece o sistema que está sendo controlado. Para ver isso, considere uma situação em que o erro $(y(t) - x_t)$ está mudando rapidamente com o tempo, como é o caso em nosso controlador P anterior. A derivada desse erro vai então contrariar o termo proporcional, o que reduzirá a resposta global à perturbação. Porém, se o mesmo erro persistir e não mudar, a derivada desaparecerá, e o termo proporcional dominará a escolha do controle.

A Figura 25.22(c) mostra o resultado da aplicação desse controlador PD ao nosso braço robô usando como parâmetros de ganho $K_P = 0{,}3$ e $K_D = 0{,}8$. É claro que o caminho resultante será muito mais suave e não exibirá oscilações óbvias. Como esse exemplo sugere, um termo diferencial pode tornar estável um controlador que de outra forma não seria estável.

Na prática, os controladores PD também possuem modos de falha. Em particular, os controladores PD podem deixar de regular um erro até reduzi-lo a zero, mesmo na ausência de perturbações externas. Muitas vezes, tal situação é o resultado de uma força externa sistemática que não faz parte do modelo. Um carro autônomo dirigindo em uma superfície inclinada, por exemplo, pode ficar sistematicamente puxado para um lado. O desgaste e o movimento rápido dos braços do robô causa erros sistemáticos semelhantes. Em tais situações é necessário realimentação sobreproporcional para conduzir o erro mais perto de zero. A solução para esse problema reside na adição de um terceiro termo à lei de controle, baseado no erro integrado ao longo do tempo.

$$a_t = K_P(y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t} . \qquad (25.2)$$

Aqui, K_i é outro parâmetro de ganho. A expressão $\int(y(t) - x_t)\, dt$ calcula a integral do erro ao longo do tempo. O efeito desse termo é que desvios duradouros entre o sinal de referência e o estado real são corrigidos. Por exemplo, se x_t for menor que $y(t)$ por um longo período de tempo, essa integral crescerá até o controle resultante a_t forçar o erro a se reduzir. Assim, termos integrais asseguram que um controlador não exibirá erros sistemáticos a expensas de um perigo maior de comportamento oscilatório. Um controlador com todos os três termos é chamado **controlador PID**. Os controladores PID são amplamente utilizados na indústria, para uma variedade de problemas de controle.

25.6.2 Controle de campo potencial

Introduzimos campos potenciais como função de custo adicional no planejamento de movimento de robôs, mas eles também podem ser usados para gerar o movimento do robô diretamente, dispensando por completo a fase de planejamento de caminho. Para conseguir isso, temos de definir uma força de atração que puxe o robô em direção à sua configuração de objetivo e um campo potencial de repulsão que empurre o robô para longe dos obstáculos. Tal campo tão potencial é mostrado na Figura 25.23. Seu único mínimo global é a configuração de destino, e o valor é a soma da distância até essa configuração de destino com a proximidade a obstáculos. Nenhum planejamento foi envolvido na geração do campo potencial mostrado na figura. Devido a isso, os campos potenciais são adequados para controle em tempo real. A Figura 25.23(a) mostra duas trajetórias de um robô que executa a subida da colina no campo potencial, sob duas configurações iniciais diferentes. Em muitas aplicações, o campo potencial pode ser calculado de modo eficiente para qualquer configuração dada. Além disso, a otimização do potencial corresponde a calcular o gradiente do potencial para a configuração atual do robô. Em geral, esses cálculos são extremamente eficientes, em especial quando comparados a algoritmos de planejamento de caminho, todos exponenciais na dimensionalidade do espaço de configuração (os GDLs) no pior caso.

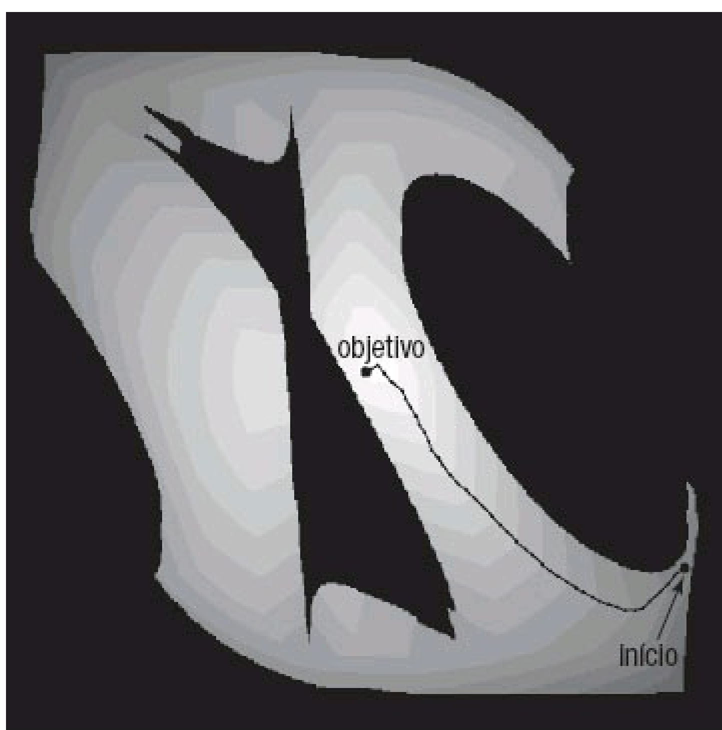




Figura 25.23 Controle de campo potencial. O robô percorre no sentido ascendente um campo potencial composto por forças de repulsão exercidas a partir dos obstáculos e por uma força de atração que corresponde à configuração de destino. (a) Caminho bem-sucedido. (b) Ótimo local.

O fato de a abordagem de campo potencial conseguir encontrar um caminho até o objetivo com tanta eficiência, mesmo em longas distâncias no espaço de configuração, levanta a questão de saber se, afinal, existe necessidade de planejamento em robótica. As técnicas de campo potencial são suficientes ou simplesmente tivemos sorte em nosso exemplo? A resposta é que de fato tivemos sorte. Os campos potenciais têm muitos mínimos locais que podem gerar armadilhas para o robô. Na Figura 25.23(b), o robô se aproxima do obstáculo simplesmente girando a articulação de seu ombro até ficar paralisado no lado errado do obstáculo. O campo potencial não é rico o suficiente para fazer o robô dobrar o cotovelo, de modo que o braço fique abaixo do obstáculo. Em outras palavras, as técnicas de campo potencial são ótimas para controle local de robôs, mas ainda exigem planejamento global. Outra desvantagem importante dos campos potenciais é que as forças que eles geram dependem apenas das posições do obstáculo e do robô, não da velocidade do robô. Desse modo, o controle de campo potencial é realmente um método cinemático e pode falhar se o robô estiver se movendo com rapidez.

25.6.3 Controle reativo

Até agora, consideramos decisões de controle que exigem algum modelo do ambiente para construir um caminho de referência ou um campo potencial. Existem algumas dificuldades com essa abordagem. Primeiro, modelos que são suficientemente precisos com frequência são difíceis de obter, em especial em ambientes complexos ou remotos, como a superfície de Marte. Em segundo lugar, mesmo em casos em que podemos criar um modelo com exatidão suficiente, as dificuldades computacionais e o erro de localização podem tornar essas técnicas impraticáveis. Em alguns casos,

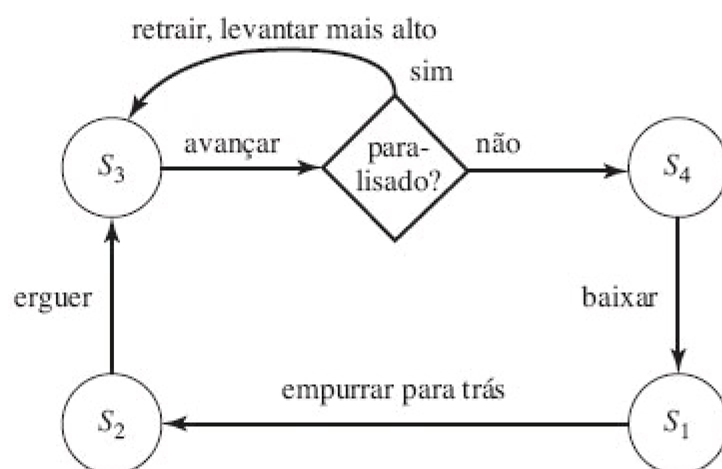
um projeto de agente de reflexo — chamado **controle reativo** — é mais apropriado.

Por exemplo, considere um robô com pernas tentando levantar uma perna sobre um obstáculo. Poderíamos fornecer uma regra para esse robô que lhe diga para levantar a perna a uma altura *h* pequena e movê-la para a frente e, se a perna encontrar um obstáculo, movê-la de volta e começar de novo com uma altura *h*. Você poderia afirmar que *h* está modelando um aspecto do mundo, mas também podemos pensar em *h* como uma variável auxiliar do controlador do robô, desprovido de significado físico direto.

Um exemplo desse tipo é o robô de seis pernas (hexápode), mostrado na Figura 25.24(a), que tem a tarefa de caminhar por terreno acidentado. Os sensores do robô são grosseiramente inadequados para se obter modelos do terreno com exatidão suficiente para fazer funcionar qualquer uma das técnicas de planejamento de caminho descritas na seção anterior. Além disso, ainda que adicionássemos sensores suficientemente precisos, os 12 graus de liberdade (dois para cada perna) tornariam o problema de planejamento de caminho resultante computacionalmente intratável.



(a)



(b)

Figura 25.24 (a) Um robô hexápode. (b) Máquina de estados finitos ampliada (MEFA) para o controle de uma única perna. Note que essa MEFA reage ao *feedback* de sensores: se uma perna ficar paralisada durante a fase de oscilação para a frente, ela será erguida a uma altura cada vez maior.

Contudo, é possível especificar um controlador diretamente, sem um modelo ambiental explícito. (Já vimos isso no caso do controlador PD, que era capaz de manter um braço robô complexo no destino *sem* um modelo explícito da dinâmica do robô; porém, ele exige um caminho de referência gerado a partir de um modelo cinemático.) Para o robô hexápode, escolhemos primeiro uma **marcha** ou padrão de movimento dos membros. Uma marcha estaticamente estável é mover primeiro a parte da frente direita, a direita de trás e as pernas do centro à esquerda para a frente (mantendo as outras três fixas) e, em seguida, mover as outras três. Essa marcha funciona muito bem em terreno plano. Em terreno acidentado, os obstáculos podem impedir que as pernas oscilem para a frente. Esse problema pode ser superado por uma regra de controle notavelmente simples: *quando o movimento de avanço de uma perna for bloqueado, simplesmente retraia a perna, levante-a a uma altura maior e tente outra vez*. O controlador resultante é mostrado na Figura 25.24(b) como uma máquina de estados finitos; ele constitui um agente de reflexo com estado, onde o estado interno é representado pelo índice do estado de máquina atual (de s_1 até s_4).

As variantes desse controlador simples orientado pelo *feedback* foram criadas para gerar um padrão de caminhada notavelmente robusto, capaz de manobrar o robô em um terreno acidentado. É claro que tal controlador é livre de modelos e não delibera ou utiliza a busca para gerar controles. Quando se executa tal controlador, a realimentação do ambiente desempenha um papel crucial no comportamento gerado pelo robô. Sozinho, o software não especifica o que realmente acontecerá quando o robô for colocado em um ambiente. O comportamento que emerge da interação de um controlador (simples) e um ambiente (complexo) com frequência é chamado **comportamento emergente**. No sentido exato, todos os robôs descritos neste capítulo exibem comportamento emergente, devido ao fato de que nenhum modelo é perfeito. Porém, historicamente, o termo foi reservado para indicar técnicas de controle que não utilizam modelos ambientais explícitos. O comportamento emergente também é característico de grande número de organismos biológicos.

25.6.4 Controle de aprendizagem por reforço

Uma forma particularmente interessante de controle baseia-se na forma de **busca de política** de aprendizagem por reforço (veja a Seção 21.5). Esse trabalho tem sido muito influente nos últimos anos, pois resolveu problemas desafiadores de robótica para os quais não existia solução anteriormente. Um exemplo é o voo de helicóptero autônomo acrobático. A Figura 25.25 mostra a cambalhota de um pequeno helicóptero autônomo controlado por rádio. Essa manobra é um desafio devido à natureza altamente não linear de aerodinâmica envolvida. Apenas os mais experientes pilotos humanos são capazes de realizá-la. No entanto, o método de busca de política (como descrito no Capítulo 21), utilizando apenas poucos minutos de computação, aprendeu uma política que pode executar cambalhotas com segurança.



Figura 25.25 Múltiplas exposições de um helicóptero RC executando uma cambalhota com base em política aprendida com aprendizagem por reforço. Imagens cedidas por Andrew Ng, da Universidade de Stanford.

A pesquisa pela melhor política precisa de um modelo exato do domínio antes que possa encontrar uma política. A entrada para esse modelo é o estado do helicóptero no tempo t, os controles no tempo, e o estado resultante no tempo t + Dt. O estado de um helicóptero pode ser descrito pela coordenada 3-D do veículo, seus ângulos de guinada, arremesso e movimento de rotação, e a taxa de variação dessas seis variáveis. Os controles são os controles manuais do helicóptero: acelerador, arremesso, elevador, profundor e leme de direção. Tudo o que resta é o estado resultante — como é vamos definir um modelo que informa exatamente como o helicóptero responde a cada controle? A resposta é simples: deixe um piloto humano especialista pilotar o helicóptero e registre os controles que o especialista transmite pelo rádio e as variáveis de estado do helicóptero. Cerca de quatro

minutos de voo humano controlado são suficientes para construir um modelo preditivo que seja suficientemente preciso para simular o veículo.

O que é notável sobre esse exemplo é a facilidade com que essa abordagem de aprendizagem resolve um problema desafiador de robótica. Esse é um dos muitos sucessos de aprendizagem de máquina nas áreas científicas anteriormente dominadas por análise matemática e modelagem cuidadosa.

25.7 ARQUITETURAS DE SOFTWARE PARA ROBÓTICA

Uma metodologia para estruturar algoritmos é chamada **arquitetura de software**. Em geral, uma arquitetura inclui linguagens e ferramentas para escrever programas, bem como uma filosofia global sobre como os programas podem ser reunidos.

As arquiteturas de software modernas para robótica devem decidir como combinar controle reativo e controle deliberativo baseado em modelos. Em vários aspectos, o controle reativo e o controle deliberativo têm pontos fortes e deficiências ortogonais. O controle reativo é orientado para sensores e é apropriado para a tomada de decisões de baixo nível em tempo real. Porém, o controle reativo raramente gera uma solução plausível no nível global porque as decisões de controle globais dependem de informações que não podem ser percebidas no instante da tomada de decisões. Para tais problemas, o controle deliberativo é mais apropriado.

Consequentemente, a maioria das arquiteturas de robôs utiliza técnicas reativas nos níveis mais baixos de controle, com técnicas deliberativas nos níveis mais altos. Encontramos tal combinação em nossa discussão de controladores PD, onde combinamos um controlador PD (reativo) com um planejador de caminho (deliberativo). Arquiteturas que combinam técnicas reativas e deliberativas costumam ser chamadas de **arquiteturas híbridas**.

25.7.1 Arquitetura de subsunção

A **arquitetura de subsunção** (Brooks, 1986) é uma estrutura para montagem de controladores reativos a partir de máquinas de estados finitos. Os nós nessas máquinas podem conter testes para certas variáveis de sensores e, nesse caso, o rastreio da execução de uma máquina de estados finitos estará condicionado ao resultado de tal teste. Os arcos podem ser marcados com mensagens que serão geradas quando eles forem percorridos e que serão enviadas aos motores do robô ou a outras máquinas de estados finitos. Além disso, as máquinas de estados finitos têm relógios internos que controlam o tempo de duração do percurso de um arco. As máquinas resultantes normalmente são referidas como **máquinas de estados finitos ampliadas**, ou MEFAs, nas quais a ampliação se refere ao uso de relógios internos.

Um exemplo de MEFA simples é a máquina de quatro estados mostrada na Figura 25.24(b), que gera o movimento cíclico das pernas de um andarilho hexápode. Essa MEFA implementa um controlador cíclico cuja execução, de modo geral, não depende de realimentação ambiental. No entanto, a fase de avanço depende da realimentação do sensor. Se a perna ficar paralisada,

significando que deixou de executar a oscilação para a frente, o robô vaiá retrair a perna, erguê-la um pouco mais alto e tentar executar a oscilação para a frente mais uma vez. Desse modo, o controlador poderá *reagir* a contingências que surgirem da interação entre o robô e seu ambiente.

A arquitetura de subsunção oferece primitivas adicionais para sincronizar MEFAs e para combinar valores de saída de várias MEFAs possivelmente conflitantes. Desse modo, ela permite ao programador compor controladores cada vez mais complexos de forma bottom-up. Em nosso exemplo, poderíamos começar com MEFAs para pernas individuais, seguidas por uma MEFA para coordenar várias pernas. Sobre tudo isso, poderíamos implementar comportamentos de nível mais altos, como evitar colisões, o que poderia envolver a ação de voltar e virar-se (mudar de direção).

A ideia de compor controladores de robôs a partir de MEFAs é intrigante. Imagine o quanto seria difícil gerar o mesmo comportamento com qualquer dos algoritmos de planejamento de caminho do espaço de configuração descritos na seção anterior. Primeiro, precisaríamos de um modelo preciso do terreno. O espaço de configuração de um robô com seis pernas, cada uma das quais é comandada por dois motores independentes, totaliza 18 dimensões (12 dimensões para a configuração das pernas e seis para a posição e a orientação do robô em relação ao seu ambiente). Mesmo que nossos computadores fossem rápidos o bastante para encontrar caminhos em espaços com números de dimensão tão altos, teríamos de nos preocupar com efeitos prejudiciais como a possibilidade de o robô escorregar em um declive. Devido a tais efeitos estocásticos, um único caminho pelo espaço de configuração quase certamente seria muito frágil, e mesmo um controlador PID talvez não fosse capaz de lidar com tais contingências. Em outras palavras, a geração deliberada de um comportamento de movimento representa simplesmente um problema complexo demais para os algoritmos atuais de planejamento de movimento de robôs.

Infelizmente, a arquitetura de subsunção tem seus próprios problemas. Primeiro, as MEFAs normalmente são orientadas pela entrada bruta dos sensores, uma organização que funciona se os dados dos sensores forem confiáveis e contiverem todas as informações necessárias para a tomada de decisões, mas que falha se os dados de sensores tiverem de ser integrados de modos não triviais ao longo do tempo. Portanto, os controladores de subsunção são aplicados principalmente a tarefas locais, como acompanhar uma parede ou mover-se em direção a fontes de luz visível. Em segundo lugar, a falta de deliberação dificulta a mudança da tarefa do robô. Em geral, um robô de subsunção executa apenas uma tarefa e não tem nenhuma noção de como modificar seus controles para acomodar objetivos de controle diferentes (da mesma forma que o besouro de esterco que vimos na página 39). Por fim, os controladores de subsunção tendem a ser difíceis de entender. Na prática, a complicada interação entre dezenas de MEFAs (e entre elas e o ambiente) está além daquilo que a maioria dos programadores humanos pode compreender. Por todas essas razões, a arquitetura de subsunção raramente é usada em robótica comercial, apesar de sua grande importância histórica. Porém, teve influência sobre outras arquiteturas e sobre componentes individuais de algumas arquiteturas.

25.7.2 Arquitetura de três camadas

As arquiteturas híbridas combinam reação e deliberação. Sem dúvida, a arquitetura híbrida mais

popular é a **arquitetura de três camadas**, que consiste em uma camada reativa, uma camada executiva e uma camada deliberativa.

A **camada reativa** fornece controle de baixo nível ao robô. Ela se caracteriza por um laço repetitivo compacto de sensor-ação. Seu ciclo de decisão frequentemente é da ordem de milissegundos.

A **camada executiva** (ou camada de sequenciamento) serve como o fator de união entre a camada reativa e a camada deliberativa. Ela aceita diretivas emitidas pela camada deliberativa e as encaminha para a camada reativa. Por exemplo, a camada executiva poderia manipular um conjunto de pontos gerados por um planejador de caminho deliberativo e tomar decisões como qual comportamento reativo invocar. Os ciclos de decisão na camada executiva em geral são da ordem de um segundo. Acamada executiva também é responsável pela integração de informações de sensores em uma representação do estado interno. Por exemplo, ela pode hospedar as rotinas de localização e de mapeamento on-line do robô.

A **camada deliberativa** gera soluções globais para tarefas complexas usando o planejamento. Devido à complexidade computacional envolvida na geração de tais soluções, seu ciclo de decisão com frequência é da ordem de minutos. A camada deliberativa (ou camada de planejamento) utiliza modelos para tomada de decisões. Esses modelos podem ser fornecidos previamente ou aprendidos a partir dos dados, e em geral utilizam informações de estados obtidas na camada executiva.

As variantes da arquitetura de três camadas podem ser encontradas na maioria dos sistemas de software de robôs modernos. A decomposição em três camadas não é muito rígida. Alguns sistemas de software de robôs possuem camadas adicionais, como camadas de interface do usuário que controlam a interação com pessoas ou camadas responsáveis pela coordenação das ações de um robô com as de outros robôs que operam no mesmo ambiente.

25.7.3 Arquitetura de pipeline

Outra arquitetura para robôs é conhecida como **arquitetura de pipeline**. Assim como a arquitetura de subsunção, a arquitetura de pipeline executa múltiplos processos em paralelo. No entanto, nessa arquitetura os módulos específicos assemelham-se aos da arquitetura de três camadas.

A Figura 25.26 mostra um exemplo de arquitetura de pipeline, que é usado para controlar um carro autônomo. Os dados entram nesse pipeline na **camada de interface de sensor**. A camada de percepção então atualiza os modelos internos do robô do meio ambiente com base nesses dados. Em seguida, esses modelos são entregues para o **planejamento e para a camada de controle**, que ajustam os planos internos do robô para que se tornem controles efetivos do robô. São, então, comunicados de volta ao veículo através da **camada de interface do veículo**.

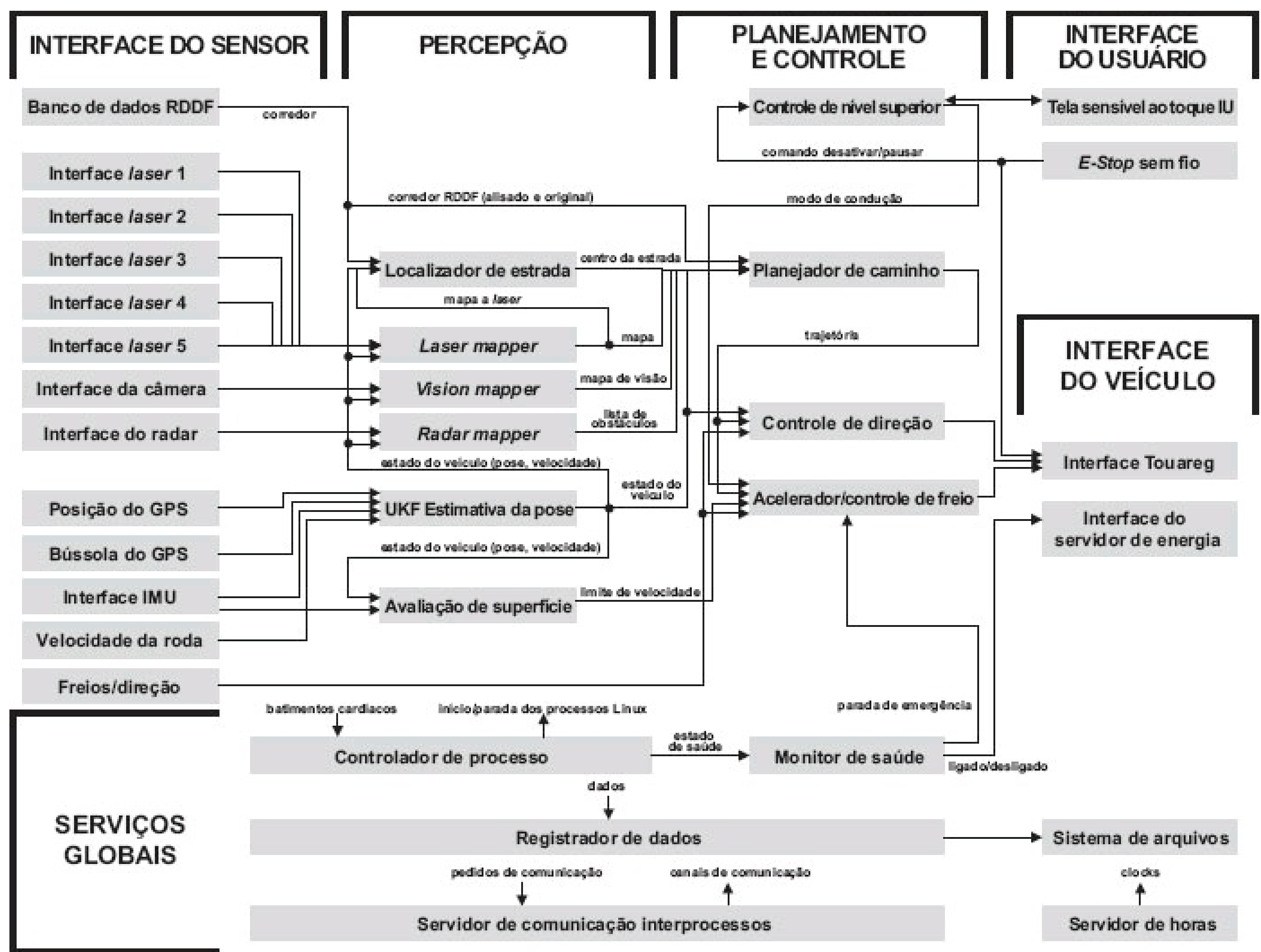


Figura 25.26 Arquitetura de software de um carro robô. Esse software implementa dados pipeline, em que todos os módulos processam os dados simultaneamente.

A chave da arquitetura de pipeline é que tudo isso acontece em paralelo. Enquanto a camada de percepção processa os dados do sensor mais recentes, a camada de controle baseia suas escolhas em dados ligeiramente mais velhos. Dessa forma, a arquitetura de pipeline é semelhante ao cérebro humano. Não desligamos nossos controladores de movimento ao digerir dados sensoriais novos. Em vez disso, percebemos, planejamos e agimos; tudo ao mesmo tempo. Os processos na arquitetura de pipeline são executados de forma assíncrona, e os cálculos são baseados em dados. O sistema resultante é robusto e rápido.

A arquitetura na Figura 25.26 também contém outros módulos transversais responsáveis por estabelecer a comunicação entre os diferentes elementos do gasoduto.

25.8 DOMÍNIOS DE APLICAÇÃO

Agora, listaremos alguns dos domínios de aplicação da tecnologia de robótica.

Indústria e agricultura. Tradicionalmente, os robôs têm sido utilizados em áreas que exigem trabalho difícil para humanos, ainda que sejam estruturadas o suficiente para serem adequadas à automação robótica. O melhor exemplo é a linha de montagem, onde os manipuladores executam

rotineiramente tarefas como montagem, substituição de peças, manipulação de materiais, soldagem e pintura. Em muitas dessas tarefas, os robôs se tornaram mais econômicos que trabalhadores humanos. Em ambientes externos, muitas máquinas pesadas que utilizamos na colheita, em minas ou em escavações do solo foram transformadas em robôs. Por exemplo, um projeto recente em Carnegie Mellon demonstrou que os robôs podem raspar a tinta de grandes embarcações cerca de 50 vezes mais rápido que as pessoas, e com impacto ambiental muito reduzido. Os protótipos de robôs autônomos para mineração têm demonstrado ser mais velozes e mais precisos que as pessoas no transporte de minérios em minas subterrâneas. Os robôs são usados para gerar mapas de alta precisão de minas abandonadas e sistemas de esgotos. Embora muitos desses sistemas ainda estejam em suas fases de protótipo, é apenas uma questão de tempo a substituição por robôs em grande parte do trabalho semimecânico executado atualmente por seres humanos.

Transporte. O transporte robótico tem muitas facetas: desde helicópteros autônomos que entregam objetos em locais de difícil acesso por outros meios até cadeiras de roda automáticas que transportam pessoas incapazes de controlar por si mesmas esses veículos e até entregadores autônomos que superam motoristas humanos experientes no transporte de contêineres de navios até caminhões em docas de carga. Um exemplo simples de robôs para transporte em recintos fechados, ou *gofers*, é o robô Helpmate mostrado na Figura 25.27(a). Esse robô foi distribuído em dezenas de hospitais para transportar alimentos e outros itens. Os pesquisadores desenvolveram sistemas robóticos semelhantes a automóveis que podem navegar de forma autônoma em autoestradas ou em outros tipos de terrenos. Em instalações industriais, veículos autônomos são empregados habitualmente para transportar mercadorias em depósitos e entre as linhas de produção. O sistema Kiva, mostrado na Figura 25.27(b), ajuda os trabalhadores em centros de realização de embalagem de bens para contêineres de navios.



(a)



(b)

Figura 25.27 (a) O robô Helpmate transporta alimentos e outros artigos médicos em dezenas de hospitais em todo o mundo. (b) Os robôs Kiva fazem parte de um sistema de manuseio de materiais para mover prateleiras em centros de atendimento. Imagem cedida por Sistemas Kiva.

Muitos desses robôs exigem modificações ambientais para sua operação. As modificações mais comuns são recursos auxiliares de localização como ciclos indutivos no piso, balizas ativas, etiquetas de código de barras e satélites GPS. Um desafio aberto em robótica é o projeto de robôs que possam utilizar indicações naturais, em vez de dispositivos artificiais para navegar, em particular em ambientes como o oceano profundo em que o GPS não está disponível.

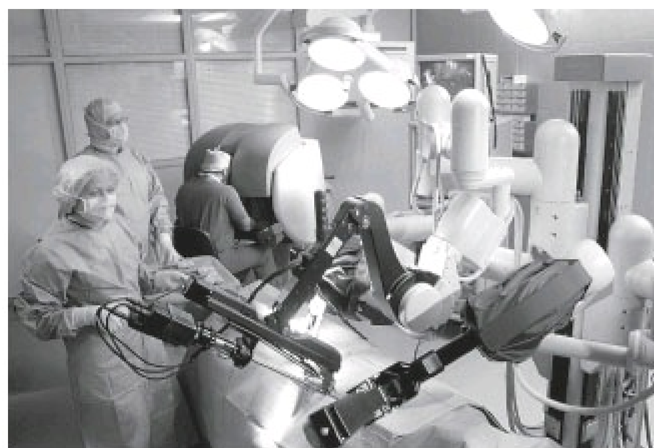
Carros robóticos. A maioria das pessoas utiliza carros todos os dias. Muitos de nós fazemos chamadas de telefone celular enquanto dirigimos.

Alguns de nós até enviamos mensagens de texto. O triste resultado é que mais de um milhão de pessoas morrem a cada ano em acidentes de trânsito. Carros robóticos como o BOSS e o STANLEY oferecem esperança: não só vão tornar a condução muito mais segura, mas também vão nos livrar da necessidade de prestar atenção à estrada durante nosso deslocamento diário.

O progresso em carros robóticos foi estimulado pelo DARPA Grand Challenge, uma corrida com mais de 160 quilômetros de terreno deserto não ensaiado, o que representou uma tarefa muito mais desafiadora do que já havia sido realizado antes. O veículo STANLEY de Stanford completou o curso em menos de sete horas em 2005, ganhando um prêmio de dois milhões de dólares e um lugar no National Museum of American History. A Figura 25.28(a) mostra o BOSS, que em 2007 venceu o DARPA Urban Challenge, uma corrida complicada nas ruas da cidade onde os robôs enfrentaram outros robôs e tiveram que obedecer às regras de trânsito.



(a)



(b)

Figura 25.28 (a) Carro robótico BOSS, que ganhou o DARPA Urban Challenge. Cedido por Carnegie Mellon University. (b) Robôs cirúrgicos na sala de cirurgia. Imagem cedida por Vinci Surgical Systems.

Cuidados com a saúde. Os robôs são cada vez mais usados para auxiliar os cirurgiões no posicionamento de instrumentos durante operações em órgãos complicados como o cérebro, os olhos e o coração. A Figura 25.28(b) mostra um sistema desse tipo. Os robôs se tornaram ferramentas indispensáveis em certos tipos de cirurgias de reposição da bacia, graças à sua elevada precisão. Em estudos-piloto, dispositivos robóticos são usados para reduzir o perigo de lesões durante a realização de exames delicados do cólon. Fora do ambiente cirúrgico, os pesquisadores começaram a desenvolver auxiliares robóticos para pessoas em idade avançada e portadoras de deficiência, como andarilhos robóticos inteligentes e brinquedos inteligentes que lembram a hora da medicação e proporcionam conforto. Os pesquisadores também estão trabalhando em dispositivos robóticos para

reabilitação que ajuda as pessoas na realização de determinados exercícios.

Ambientes perigosos. Os robôs têm ajudado as pessoas na limpeza de resíduos nucleares, mais notavelmente em Chernobyl e Three Mile Island. Os robôs estavam presentes após o colapso do World Trade Center, em que adentraram estruturas consideradas muito perigosas para busca humana e equipes de resgate.

Alguns países usam robôs para transportar munição e para desarmar bombas — uma tarefa notoriamente perigosa. Protótipos de robôs para limpar campos minados, em terra e no mar, estão sendo desenvolvidos por vários projetos de pesquisa. A maioria dos robôs existentes para essas tarefas são teleoperados — um ser humano opera por controle remoto. O fornecimento de tais robôs com autonomia será o próximo passo importante.

Exploração. Os robôs chegaram aonde ninguém chegou antes, incluindo a superfície de Marte (veja a Figura 25.2(b) e a capa). Braços robóticos ajudam os astronautas na implantação e recuperação de satélites e na construção da Estação Espacial Internacional. Os robôs também ajudam a explorar sob o mar. São utilizados rotineiramente para adquirir mapas de navios afundados. A Figura 25.29 mostra um robô mapeando uma mina de carvão abandonada, juntamente com um modelo 3-D da mina adquirido através do uso de sensores de alcance. Em 1996, uma equipe de pesquisas colocou um robô com pernas dentro da cratera de um vulcão ativo para adquirir dados sobre pesquisa climática. Veículos aéreos não tripulados conhecidos como **drones** são usados em operações militares. Os robôs estão se tornando ferramentas muito eficazes para juntar informações em domínios que são difíceis (ou perigosos) ao acesso das pessoas.


(a)

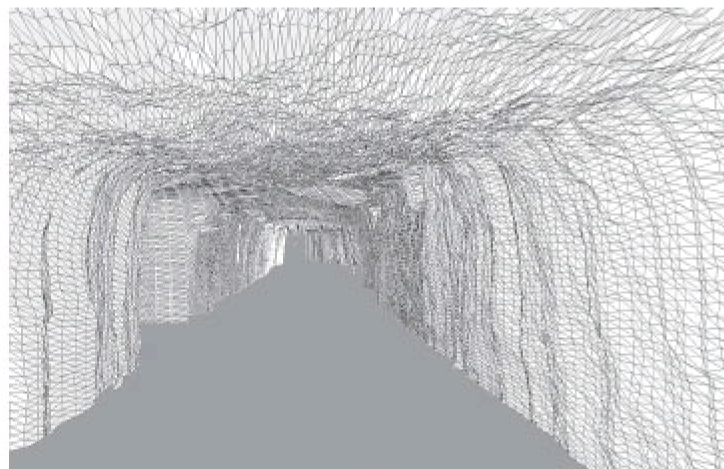

(b)

Figura 25.29 (a) Robô mapeando uma mina de carvão abandonada. (b) Mapa 3-D da mina obtido pelo robô.

Serviços pessoais. O serviço é um dos próximos domínios de aplicação da robótica. Os robôs serviçais prestam assistência a indivíduos na realização de tarefas do dia a dia. Os robôs para serviços domésticos comercialmente disponíveis incluem aspiradores de pó autônomos, cortadores de grama e os *caddies* em partidas de golfe. O robô móvel mais popular do mundo é um robô de serviço personalizado: o aspirador de pó robótico **Roomba**, mostrado na Figura 25.30(a). Mais de três milhões de Roombas foram vendidos. O Roomba pode navegar de forma autônoma e executar suas tarefas sem ajuda humana.



(a)

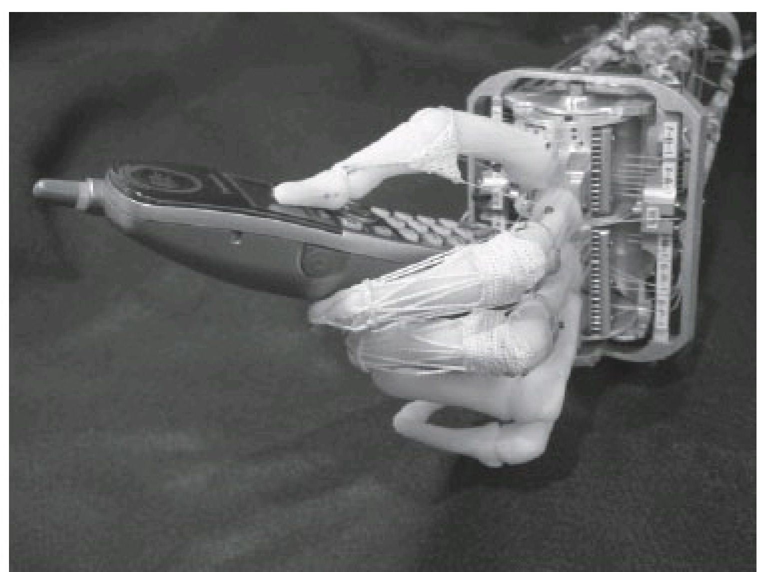

(b)

Figura 25.30 (a) Roomba, o robô móvel mais vendido do mundo, aspirador de pisos. Imagem cedida por iRobot, © 2009. (b) Mão robótica modelada a partir de uma mão humana. Imagem cedida pela University of Washington e pela Carnegie Mellon University.

Alguns robôs de serviços operam em lugares públicos, como os quiosques robóticos de informações que estão distribuídos em centros comerciais e feiras de negócios ou em museus como guias de excursões. As tarefas de serviços exigem interação humana e a habilidade para lidar com ambientes imprevisíveis e dinâmicos.

Entretenimento. Os robôs começaram a conquistar a indústria de entretenimento e brinquedos. Na Figura 25.6(b) vemos o **futebol robótico**, um jogo competitivo muito semelhante ao futebol humano, mas jogado com robôs móveis autônomos. O futebol robótico oferece excelentes oportunidades para pesquisa em IA, pois levanta grande variedade de problemas que servem de protótipos para muitas outras aplicações mais sérias dos robôs. As competições anuais de futebol robótico atraem grande número de pesquisadores de IA e aumentaram bastante o interesse para o campo da robótica.

Ampliação da capacidade humana. Um último domínio da aplicação da tecnologia robótica é a ampliação da capacidade humana. Os pesquisadores desenvolveram máquinas andarilhas dotadas de pernas que podem transportar pessoas, de maneira muito semelhante a uma cadeira de rodas. Atualmente, um grande esforço de pesquisa se concentra no desenvolvimento de dispositivos que facilitam a vida das pessoas, ajudando-as a caminhar ou mover os braços, oferecendo forças adicionais através de acessórios extracorpóreos. Se tais dispositivos forem permanentemente conectados, eles poderão ser considerados membros robóticos artificiais. A Figura 25.30(b) mostra uma mão robótica que pode servir como dispositivo protético no futuro.

A teleoperação robótica, ou telepresença, é outra forma de ampliação da capacidade humana. A teleoperação envolve a execução de tarefas a longas distâncias, com o auxílio de dispositivos robóticos. Uma configuração popular para teleoperação robótica é a configuração de mestre-escravo, em que um manipulador robótico emula o movimento de um operador humano remoto, medido por meio de uma interface haptotrópica. Os veículos subaquáticos muitas vezes são teleoperados; eles podem ir a uma profundidade perigosa para os humanos, mas ainda podem ser conduzidos pelo operador humano. Todos esses sistemas aumentam a habilidade das pessoas para interagirem com

seus ambientes. Alguns projetos chegam a replicar os seres humanos, pelo menos em um nível muito superficial. Os robôs humanoides estão disponíveis comercialmente em diversas empresas no Japão.

25.9 RESUMO

A robótica se preocupa com agentes inteligentes que manipulam o mundo físico. Neste capítulo, aprendemos os princípios fundamentais de hardware e software de robôs relacionados a seguir.

- Os robôs estão equipados com **sensores** que lhes permitem perceber seu ambiente e com efetuadores que eles podem utilizar para exercer forças físicas sobre esse ambiente. Em sua maioria, os robôs são manipuladores fixos (ancorados em posições fixas) ou robôs móveis que podem se deslocar pelo ambiente.
- A percepção robótica se refere à avaliação de quantidades relevantes para as decisões tomadas a partir dos dados de sensores. Para fazer isso, precisamos de uma representação interna e de um método para atualizar essa representação interna com o passar do tempo. Os exemplos comuns de problemas difíceis de percepção incluem **localização**, **mapeamento** e **reconhecimento de objetos**.
- Os **algoritmos de filtragem probabilística**, como os filtros de Kalman e os filtros de partículas, são úteis para a percepção de robôs. Essas técnicas mantêm o estado de crença, isto é, uma distribuição posterior sobre variáveis de estados.
- O planejamento do movimento dos robôs é feito normalmente no **espaço de configuração**, onde cada ponto especifica a posição e a orientação do robô e dos ângulos de suas articulações.
- Os algoritmos de busca dos espaços de configuração incluem **técnicas de decomposição** em células que decompõem o espaço de todas as configurações em um número finito de células e também técnicas de **esqueletização**, que projetam espaços de configuração sobre cópias de dimensões mais baixas. O problema de planejamento do movimento é resolvido então com o emprego de busca sobre essas estruturas mais simples.
- Um caminho encontrado por um algoritmo de busca pode ser percorrido com o uso do caminho como trajetória para um **controlador PID**. Em robótica, os controladores são necessários para acomodar pequenas perturbações; o planejamento de caminho sozinho geralmente é ineficiente.
- As técnicas de **campo potencial** realizam a navegação de robôs por funções de potencial, definidas sobre a distância até obstáculos e sobre a posição de destino. As técnicas de campo potencial podem ficar paralisadas em mínimos locais, mas podem gerar movimento diretamente, sem necessidade de planejamento do caminho.
- Às vezes, é mais fácil especificar um controlador de robô diretamente, em vez de derivar um caminho a partir de um modelo explícito do ambiente. Com frequência, tais controladores podem ser descritos como **máquinas de estados finitos** simples.
- Existem diferentes arquiteturas de projeto de software. A **arquitetura de subsunção** permite aos programadores compor controladores de robô a partir de máquinas de estados finitos interconectadas. **Arquiteturas de três camadas** são estruturas comuns para o desenvolvimento de software de robôs que integram a deliberação, o sequenciamento de submetas e o controle. A **arquitetura de pipeline** relacionada processa os dados em paralelo, através de uma sequência

de módulos correspondentes à percepção, modelagem, planejamento, controle e interfaces de robô.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

A palavra **robô** foi popularizada pelo dramaturgo tcheco Karel Capek em sua peça de 1921, *R.U.R.* (*Rossum's Universal Robots*). Nessa peça, os robôs, que cresciam quimicamente em vez de serem construídos mecanicamente, acabaram ressentidos com seus mestres e decidiram assumir o comando de tudo. Parece que, na realidade (Glanc, 1978), foi o irmão de Capek, Josef, quem primeiro combinou as palavras tchecas "*robota*" (trabalho obrigatório) e "*robotnik*" (servo) para gerar "robô" em sua breve narrativa *Opilec*, de 1917.

A expressão *robótica* foi usada primeiro por Asimov (1950). Contudo, a robótica (com outros nomes) tem uma história muito mais longa. Na mitologia grega antiga, um homem mecânico chamado Talos supostamente foi projetado e construído por Hefesto, o deus grego da metalurgia. Autômatos maravilhosos foram construídos no século XVIII — o pato mecânico de Jacques Vaucanson de 1738 seria o primeiro exemplo —, mas os comportamentos complexos que eles exibiam eram inteiramente fixados com antecedência. O exemplo mais antigo de dispositivo semelhante a um robô programável talvez tenha sido o tear de Jacquard (1805), descrito anteriormente.

O primeiro robô comercial foi um braço robô chamado **Unimate**, abreviatura de *universal automation* (automação universal). O Unimate foi desenvolvido por Joseph Engelberger e George Devol. Em 1961, o primeiro robô Unimate foi vendido à General Motors, onde foi usado para fabricar tubos de imagem (ou cinescópios) de TV. 1961 também foi o ano em que Devol obteve a primeira patente de um robô nos Estados Unidos. Onze anos mais tarde, em 1972, a Nissan Corp. estava entre as primeiras empresas a automatizar uma linha de montagem inteira com robôs; essa linha de montagem foi desenvolvida pela Kawasaki com robôs fornecidos pela empresa Unimation, de Engelberger e Devol. Esse desenvolvimento deu início a uma importante revolução que aconteceu principalmente no Japão e nos Estados Unidos, e que ainda prossegue. A Unimation continuou em atividade e culminou com o desenvolvimento, em 1978, do robô **PUMA**, abreviatura de Programmable Universal Machine for Assembly. O robô PUMA, desenvolvido inicialmente para a General Motors, foi o padrão de fato para manipulação robótica durante as duas décadas seguintes. No momento, estima-se em um milhão o número de robôs em operação no mundo inteiro, mais da metade deles instalados no Japão.

A literatura sobre pesquisa em robótica pode ser dividida, *grosso modo*, em duas partes: robôs móveis e manipuladores estacionários. A "tartaruga" de Walter Grey, construída em 1948, pode ser considerada o primeiro robô móvel autônomo, embora seu sistema de controle não fosse programável. O robô denominado "Hopkins Beast", construído no início da década de 1960 na Johns Hopkins University, era muito mais sofisticado; ele tinha hardware de reconhecimento de padrões e conseguia reconhecer a placa de cobertura de uma tomada de energia elétrica CA padrão. Esse robô era capaz de buscar as tomadas, conectar-se a elas e recarregar suas baterias! Ainda assim, tinha um repertório limitado de habilidades. O primeiro robô móvel de uso geral foi o "Shakey", desenvolvido onde era o Stanford Research Institute (agora SRI) no final da década de 1960 (Fikes e

Nilsson, 1971; Nilsson, 1984). O Shakey foi o primeiro robô a integrar percepção, planejamento e execução, e grande parte da pesquisa subsequente em IA foi influenciada por essa notável realização. Ele aparece na capa deste livro com o líder de projeto, Charlie Rosen (1917-2002). Outros projetos influentes incluem o Stanford Cart e o CMU Rover (Moravec, 1983). Cox e Wilfong (1990) descrevem um trabalho clássico em veículos autônomos.

O campo de mapeamento robótico evoluiu a partir de duas origens distintas. A primeira corrente começou com o trabalho de Smith e Cheeseman (1986), que aplicaram filtros de Kalman ao problema de localização e mapeamento simultâneos. Esse algoritmo foi implementado primeiro por Moutarlier e Chatila (1989), e mais tarde foi estendido por Leonard e Durrant-Whyte (1992); veja Dissanayake *et al.* (2001) para uma visão geral das primeiras variações de filtragem de Kalman. A segunda corrente teve início com o desenvolvimento da representação de **grade de ocupação** para mapeamento probabilístico, que especifica a probabilidade de cada posição (x, y) estar ocupada por um obstáculo (Moravec e Elfes, 1985). Uma visão geral do estado da arte em mapeamento robótico pode ser encontrada em Thrun (2002). Kuipers e Levitt (1988) foram dois dos primeiros pesquisadores a propor o mapeamento topológico em lugar do mapeamento métrico, motivados por modelos de cognição espacial humana. Uma tese seminal por Lu e Milios (1997) reconheceu a escassez da localização simultânea e o problema de mapeamento, que deu origem ao desenvolvimento das técnicas de otimização não linear por Konolige (2004) e Montemerlo e Thrun (2004), bem como métodos hierárquicos de Bosse *et al.* (2004). Shatkay e Kaelbling (1997) e Thrun *et al*. (1998) introduziram o algoritmo EM para o campo da robótica de mapeamento de associação de dados. Uma visão geral de métodos de mapeamento probabilístico pode ser encontrada em Thrun *et al*. (2005).

As primeiras técnicas de localização de robôs móveis foram pesquisadas por Borenstein *et al*. (1996). Embora a filtragem de Kalman já fosse conhecida durante décadas como um método de localização em teoria de controle, a formulação probabilística geral do problema de localização só surgiu na literatura de IA muito mais tarde, graças ao trabalho de Tom Dean e seus colegas (Dean *et al*.,1990, 1990) e ao de Simmons e Koenig (1995). O trabalho posterior introduziu a expressão **localização de Markov**. A primeira aplicação real dessa técnica foi empreendida por Burgard *et al*. (1999), através de uma série de robôs distribuídos em museus. A localização de Monte Carlo baseada em filtros de partículas foi desenvolvida por Fox *et al*. (1999) e agora é extensamente empregada.

O **filtro de partículas Rao-Blackwellized** combina a filtragem de partículas para localização de robôs com a filtragem exata para elaboração de mapas (Murphy e Russell, 2001; Montemerlo *et al*., 2002).

O estudo de robôs manipuladores, chamados originalmente **máquinas mão-olho**, evoluiu ao longo de diretrizes bem diferentes. O primeiro esforço importante de criação de uma máquina mão-olho foi MH-1 de Heinrich Ernst, descrito em sua tese de doutorado no MIT (Ernst, 1961). O projeto Machine Intelligence em Edinburgh também demonstrou um impressionante protótipo de sistema para montagem baseada na visão, chamado FREDDY (Michie, 1972). Depois desses esforços pioneiros, um trabalho intenso se concentrou em algoritmos geométricos para problemas de planejamento de movimentos determinísticos e completamente observáveis. O caráter PSPACE-difícil do planejamento de movimentos de robôs foi mostrado em um importante ensaio de Reif (1979). A

representação do espaço de configuração se deve a Lozano-Perez (1983). Teve grande influência uma série de documentos de Schwartz e Sharir sobre problemas que eles denominaram **carregadores de piano** (Schwartz *et al*., 1987).

A decomposição recursiva em células para planejamento do espaço de configuração teve origem no trabalho de Brooks e Lozano-Perez (1985) e foi melhorada de forma significativa por Zhu e Latombe (1991). Os primeiros algoritmos de esqueletização se baseavam em diagramas de Voronoi (Rowat, 1979) e **grafos de visibilidade** (Wesley e Lozano-Perez, 1979). Guibas *et al*. (1992) desenvolveram técnicas eficientes para calcular diagramas de Voronoi de modo incremental, e Choset (1996) generalizou diagramas de Voronoi para problemas de planejamento de movimentos muito mais amplos. John Canny (1988) estabeleceu o primeiro algoritmo exponencial um para um para planejamento de movimentos. O texto seminal de Jean-Claude Latombe (1991) abrange uma variedade de enfoques para o problema de planejamento de movimentos, assim como os textos de Choset *et al*. (2004) e LaValle (2006). Kavraki *et al*. (1996) desenvolveram roteiros probabilísticos, que atualmente constituem o método mais efetivo. O planejamento de movimento ótimo com detecção limitada foi investigado por Lozano-Perez *et al*. (1984) e por Canny e Reif (1987) utilizando a ideia de incerteza de intervalos, em vez da incerteza probabilística. A navegação baseada em marcos (Lazanas e Latombe, 1992) utiliza muitas das mesmas ideias na arena de robôs móveis. Trabalhos importantes com a aplicação de métodos POMDP (Seção 17.4) para planejamento de movimento sob incerteza em robótica são devidos a Pineau *et al.* (2003) e Roy *et al*. (2005).

O controle de robôs como sistemas dinâmicos — seja para manipulação ou navegação — gerou extensa literatura, da qual o material deste capítulo é apenas um breve resumo. Os trabalhos importantes incluem uma trilogia sobre controle de impedância de Hogan (1985) e um estudo geral da dinâmica de robôs feito por Featherstone (1987). Dean e Wellman (1991) estavam entre os primeiros pesquisadores a tentar reunir a teoria de controle e os sistemas de planejamento de IA. Três livros didáticos clássicos sobre a matemática de manipulação de robôs se devem a Paul (1981), Craig (1989) e Yoshikawa (1990). O estudo da ação de **agarrar** também é importante em robótica — o problema de determinar uma garra estável é bastante difícil (Mason e Salisbury, 1985). Uma ação de agarrar competente exige sensibilidade ao toque, ou ***feedback*** **haptotrópico**, a fim de determinar forças de contato e detectar o deslizamento (Fearing e Hollerbach, 1985).

O controle de campo potencial, que tenta resolver os problemas de planejamento e controle do movimento simultaneamente, foi introduzido na literatura de robótica por Khatib (1986). Em robótica móvel, essa ideia foi vista como solução prática para o problema de evitar colisões e mais tarde foi estendida para gerar um algoritmo chamado **histogramas de campo vetorial** por Borenstein (1991). Funções de navegação, a versão de robótica de uma política de controle para processos de decisão de Markov (MDPs) determinísticos, foram introduzidas por Koditschek (1987). O reforço de aprendizagem em robótica decolou com o trabalho seminal de Bagnell e Schneider (2001) e Ng *et al.* (2004), que desenvolveram o paradigma no contexto do controle de helicópteros autônomos.

O tópico de arquiteturas de software para robôs gera um intenso debate religioso. A boa e velha candidata da IA — a arquitetura de três camadas — remonta ao projeto do robô Shakey e foi revista por Gat (1998). A arquitetura de subsunção se deve a Rodney Brooks (1986), embora ideias semelhantes tenham sido desenvolvidas independentemente por Braitenberg (1984), cujo livro, *Vehicles*, descreve uma série de robôs simples baseados na abordagem comportamental. O sucesso

do robô móvel de seis pernas de Brooks foi seguido por muitos outros projetos. Connell, em sua tese de doutorado (1989), desenvolveu um robô móvel capaz de recuperar objetos e que era inteiramente reativo. As extensões do paradigma de comportamento para sistemas de vários robôs podem ser encontradas em Mataric (1997) e Parker (1996). O GRL (Horswill, 2000) e o COLBERT (Konolige, 1997) abstraem as ideias de robótica concorrente baseada no comportamento em linguagens gerais de controle de robôs. Arkin (1998) estuda o estado da arte nesse campo.

Na última década, várias competições importantes estimularam a pesquisa sobre robótica móvel. A mais antiga competição, a competição anual de robô móvel de AAAI, começou em 1992. O primeiro vencedor da competição foi CARAMEL (Congdon *et al.,* 1992). O progresso vem sendo constante e impressionante: em competições mais recentes, os robôs entraram no complexo de conferências, encontraram o seu caminho para o balcão de registo, registaram-se para a conferência e até deram uma prosa rápida. A competição **Robocup,** lançada em 1995 pela Kitano e colaboradores (1997a), visa "desenvolver uma equipe de robôs humanoides, totalmente autônomos, que possa vencer contra o time humano campeão mundial de futebol" em 2050. O jogo ocorre em ligas de robôs simulados, robôs com rodas de tamanhos diferentes e robôs humanoides. Em 2009, equipes de 43 países participaram e o evento foi transmitido para milhões de telespectadores. Visser e Burkhard (2007) acompanharam as melhorias que foram feitas na percepção, coordenação de equipe e habilidades de baixo nível, ao longo da última década.

O **DARPA Grand Challenge**, organizado pela DARPA em 2004 e 2005, exigia que os robôs autônomos viajassem mais de 100 milhas através de terreno deserto sem ensaio em menos de 10 horas (Buehler *et al*., 2006). No evento original em 2004, nenhum robô viajou mais que 13 quilômetros, levando muitos a acreditarem que o prêmio nunca seria reivindicado. Em 2005, o robô STANLEY de Stanford ganhou a competição em pouco menos de sete horas de viagem (Thrun, 2006). A DARPA então organizou o **Urban Challenge**, uma competição na qual os robôs tinham de navegar 100 quilômetros em um ambiente urbano com tráfego. O robô BOSS da Carnegie Mellon University levou o primeiro lugar e reclamou um prêmio de $2 milhões (Urmson e Whittaker, 2008). Dickmanns e Zapp (1987) e Pomerleau (1993) são considerados como pioneiros no desenvolvimento de carros robóticos,

Dois livros iniciais, por Dudek e Jenkin (2000) e Murphy (2000), abrangem a robótica de modo geral. Uma visão mais recente é devida a Bekey (2008). Um excelente livro sobre manipulação de robô aborda tópicos avançados, como movimento compatível (Mason, 2001). Choset *et al*. (2004) e LaValle (2006) abrangeram o planejamento do movimento do robô. Thrun *et al*. (2005) forneceram uma introdução em robótica probabilística. A principal conferência sobre robótica é a IEEE International Conference on Robotics and Automation. Os periódicos líderes sobre robótica incluem o *IEEE Robotics and Automation*, o *International Journal of Robotics Research* e o *Robotics and Autonomous Systems*.

EXERCÍCIOS

25.1 A localização de Monte Carlo é *desviada* para qualquer tamanho de amostra finito — isto é, o valor esperado da posição calculado pelo algoritmo difere do valor esperado verdadeiro — devido

ao modo de funcionamento da filtragem de partículas. Nessa questão, você deverá quantificar esse desvio.

Para simplificar, considere um mundo com quatro posições possíveis do robô: $X = \{x_1, x_2, x_3, x_4\}$. Inicialmente, extraímos uniformemente $N \geq 1$ amostras dentre essas posições. Como sempre, é perfeitamente aceitável que mais de uma amostra seja gerada para qualquer das posições X. Seja Z uma variável booleana de sensor caracterizada pelas probabilidades condicionais a seguir:

$P(z \mid x_1) = 0{,}8$ $P(\neg z \mid x_1) = 0{,}2$
$P(z \mid x_2) = 0{,}4$ $P(\neg z \mid x_2) = 0{,}6$
$P(z \mid x_3) = 0{,}1$ $P(\neg z \mid x_3) = 0{,}9$
$P(z \mid x_4) = 0{,}1$ $P(\neg z \mid x_4) = 0{,}9$

A LMC utiliza essas probabilidades para gerar pesos de partículas, que são normalizados subsequentemente e usados no processo de reamostragem. Por simplicidade, vamos supor que só geramos uma nova amostra no processo de reamostragem, independentemente de N. Essa amostra poderia corresponder a qualquer uma das quatro posições em X. Desse modo, o processo de amostragem define uma distribuição de probabilidade sobre X.

a. Qual é a distribuição de probabilidade resultante sobre X para essa nova amostra? Responda a essa pergunta separadamente para $N = 1, \ldots, 10$ e para $N = \infty$.

b. A diferença entre duas distribuições de probabilidade P e Q pode ser medida pela divergência de KL, definida como:

$$KL(P, Q) = \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)} .$$

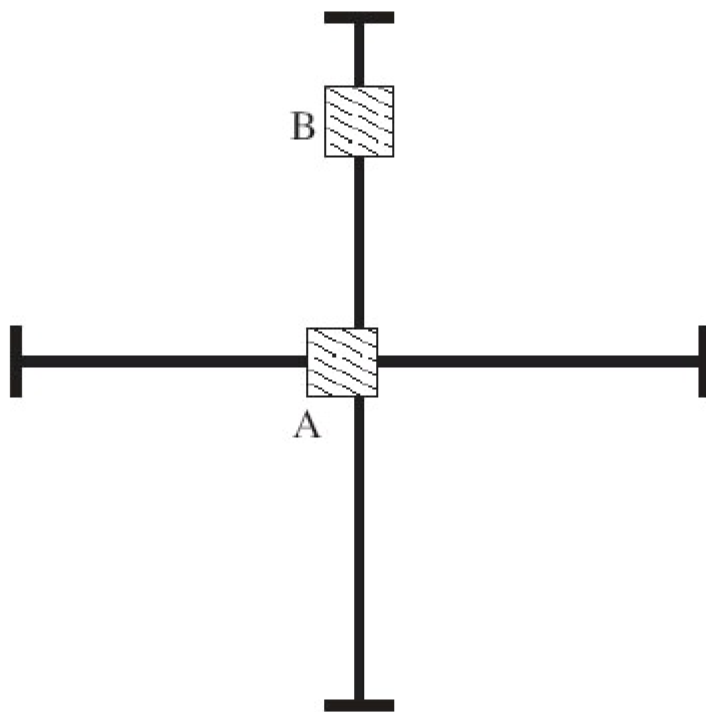
Quais são as divergências de KL entre as distribuições em (a) e a distribuição posterior verdadeira?

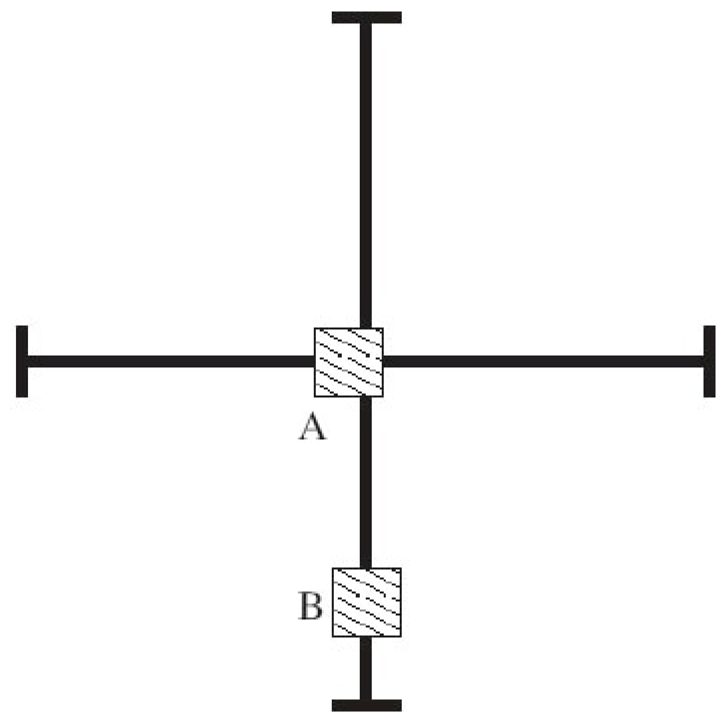
c. Que modificação da formulação do problema (não do algoritmo!) garantiria que o avaliador específico anterior fosse imparcial, mesmo para valores finitos de N? Forneça pelo menos duas dessas modificações (cada uma das quais deve ser suficiente).

25.2 Implemente a localização de Monte Carlo para um robô simulado com sensores de alcance. Um mapa de grade e dados de intervalos estão disponíveis no repositório de código em `aima.cs.berkeley.edu`. Seu exercício estará completo se você puder demonstrar uma localização global bem-sucedida do robô.

25.3 Considere um robô com dois manipuladores simples, como mostrado na Figura 25.31. O manipulador A é um bloco quadrado com duas laterais, que pode deslizar para a frente e para trás em uma haste que corre ao longo do eixo x de x = −10 a x = 10. O manipulador B é um bloco quadrado com duas laterais que pode deslizar para a frente e para trás em uma haste que corre ao longo do eixo y de y = −10 a y = 10. As varas estão fora do plano de manipulação, por isso as hastes não interferem com o movimento dos blocos. Uma configuração é então um par $\langle x, y \rangle$, onde x é a coordenada x do

centro do manipulador A e onde y é a coordenada y do centro do manipulador B. Desenhe o espaço de configuração desse robô com indicação das zonas permitidas e excluídos.





Configuração inicial <−0,5, 7> Configuração final <−0,5, −7>

Figura 25.31 Robô manipulador em duas de suas configurações possíveis.

25.4 Suponha que no Exercício 25.3 você esteja trabalhando com o robô e o problema seja encontrar um caminho a partir da configuração inicial da Figura 25.31 até a configuração final. Considere a função potencial

$$D(A, Goal)^2 + D(B, Goal)^2 + \frac{1}{D(A, B)^2}$$

onde D(A, B) é a distância entre os pontos mais próximos de A e B.

a. Mostre que a subida de encosta nesse campo potencial ficará presa em um mínimo local.

b. Descreva um campo potencial onde a subida de encosta irá resolver esse problema em particular. Não é necessário calcular os coeficientes numéricos exatos necessários, apenas a forma geral da solução. (Dica: adicione um termo que "premie" o subidor de encosta por sair do caminho de B para A, mesmo no caso em que a distância entre A e B não for reduzida no sentido acima.)

25.5 Considere o braço robô mostrado na Figura 25.14. Suponha que o elemento da base do robô tenha 60 cm de comprimento e que o braço e o antebraço tenham cada um 40 cm de comprimento. Como vimos anteriormente, a cinemática inversa de um robô com frequência não é única. Enuncie uma solução explícita de forma fechada da cinemática inversa para esse braço. Sob que condições exatas a solução é única?

25.6 Implemente um algoritmo para calcular o diagrama de Voronoi de um ambiente

bidimensional arbitrário, descrito por um array booleano $n \times n$. Ilustre seu algoritmo representando o diagrama de Voronoi para 10 mapas interessantes. Qual é a complexidade de seu algoritmo?

25.7 Este exercício explora o relacionamento entre espaço de trabalho e espaço de configuração usando os exemplos mostrados na Figura 25.32.

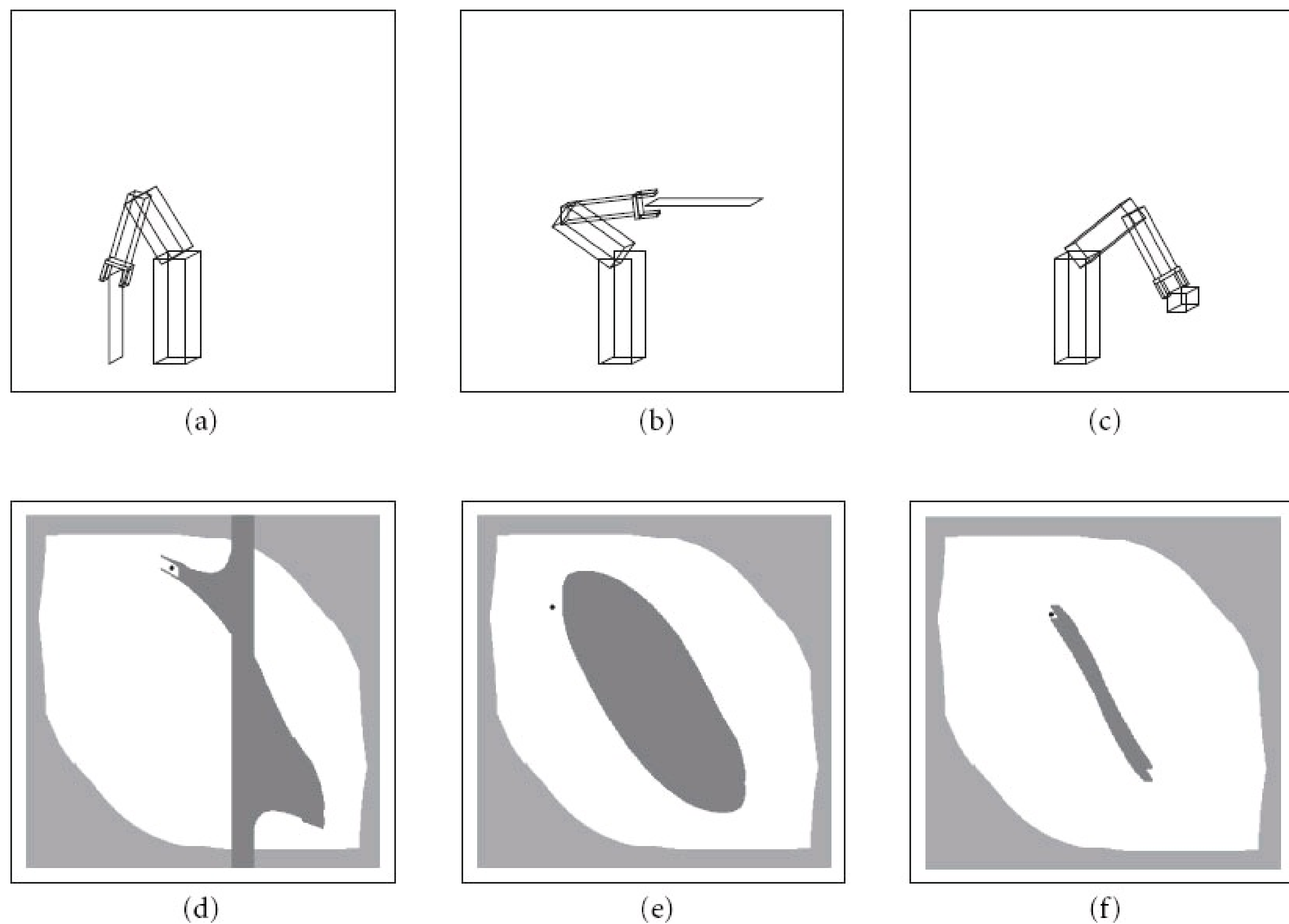


Figura 25.32 Diagramas para o Exercício 25.7.

a. Considere as configurações de robôs mostradas na Figura 25.32 de (a) até (c), ignorando o obstáculo mostrado em cada um dos diagramas. Desenhe as configurações do braço correspondentes no espaço de configuração. (*Sugestão*: Cada configuração do braço é mapeada para um único ponto no espaço de configuração, como ilustra a Figura 25.14(b).)

b. Desenhe o espaço de configuração para cada um dos diagramas de espaço de trabalho da Figura 25.32 (a)-(c). (*Sugestão*: Os espaços de configuração compartilham com o espaço mostrado na Figura 25.32(a) a região que corresponde à autocolisão, mas surgem diferenças devido à falta de inclusão de obstáculos e às diferentes posições dos obstáculos nessas figuras individuais.)

c. Para cada um dos pontos pretos na Figura 25.32(e)-(f), desenhe as configurações correspondentes do braço do robô no espaço de trabalho. Ignore as regiões sombreadas neste exercício.

d. Os espaços de configuração mostrados na Figura 25.32(e)-(f) foram gerados por um único obstáculo do espaço de trabalho (sombreamento escuro), além das limitações que surgem da restrição de autocolisão (sombreamento claro). Desenhe, para cada diagrama, o obstáculo do espaço de trabalho que corresponde à área sombreada mais escura.

e. A Figura 25.32(d) mostra que um único obstáculo planar pode decompor o espaço de trabalho em duas regiões isoladas. Qual é o número máximo de regiões isoladas entre si que podem ser criadas pela inserção de um obstáculo planar em um espaço de trabalho conectado livre de obstáculos, para um robô 2GDL? Dê um exemplo e explique por que não é possível criar nenhum número maior de regiões isoladas. E no caso de um obstáculo não planar?

25.8 Considere um robô móvel que se desloca sobre uma superfície horizontal. Suponha que o robô possa executar dois tipos de movimentos:

- Rodar para a frente uma distância especificada.
- Girar no lugar através de um ângulo especificado.

O estado de tal robô pode ser caracterizado em termos de três parâmetros $\langle x, y, \phi$, a coordenada x e a coordenada y do robô (mais precisamente, do seu centro de rotação) e a orientação do robô expressa como o ângulo da direção x positiva. A ação "*Rodar*(*D*)" tem o efeito de mudar o estado $\langle x, y, \phi$ para $\langle x + D\cos(\phi), y + D\,\mathrm{sen}(\phi), \phi\rangle$ e a ação *Girar*(θ) tem o efeito de mudar o estado $<x, y, \phi>$ para $<x, y, + \theta\,\rangle$.

a. Suponha que o robô esteja inicialmente em $<0, 0, 0>$ e, em seguida, execute as ações *Girar*(60º), *Rodar*(1), *Girar*(25º), *Rodar*(2). Qual será o estado final do robô?

b. Agora, suponha que o robô tenha controle imperfeito de sua própria rotação e que, se tentar girar de θ, pode realmente girar em qualquer ângulo entre θ – 10º e θ + 10º. Nesse caso, se o robô tenta realizar a sequência de ações em (A), há uma gama de possíveis estados finais. Quais são os valores mínimo e máximo da coordenada x, da coordenada y e a orientação no estado final?

c. Vamos modificar o modelo em (B) para um modelo probabilístico, no qual, quando o robô tenta girar de θ, seu ângulo real de rotação segue uma distribuição gaussiana com média θ e desvio-padrão de 10º. Suponha que o robô execute as ações *Girar*(90º*), Rodar*(1). Dê um argumento simples (a) para que o valor esperado da localização no final não seja igual ao resultado de girar exatamente 90º e rodar para a frente uma unidade e (b) que a distribuição de locais no final não siga uma gaussiana (não tente calcular a média verdadeira ou a distribuição verdadeira).

O ponto deste exercício é que a incerteza de rotação rapidamente dá lugar a uma porção de incerteza posicional e que lidar com a incerteza de rotação é doloroso se a incerteza for tratada em termos de intervalos rígidos ou probabilisticamente, devido ao fato de que a relação entre a orientação e a posição é ao mesmo tempo não linear e não monotônica.

25.9 Considere o robô simplificado mostrado na Figura 25.33. Suponha que as coordenadas cartesianas do robô sejam conhecidas em todos os instantes, como também as de sua posição de destino. Porém, as posições dos obstáculos são desconhecidas. O robô pode detectar obstáculos em sua proximidade imediata, como ilustra essa figura. Por simplicidade, vamos supor que o movimento

do robô seja livre de ruídos e que o espaço de estados seja discreto. A Figura 25.33 é apenas um exemplo; neste exercício, você deverá examinar todos os mundos de grades possíveis com um caminho válido desde a posição inicial até a posição objetivo.

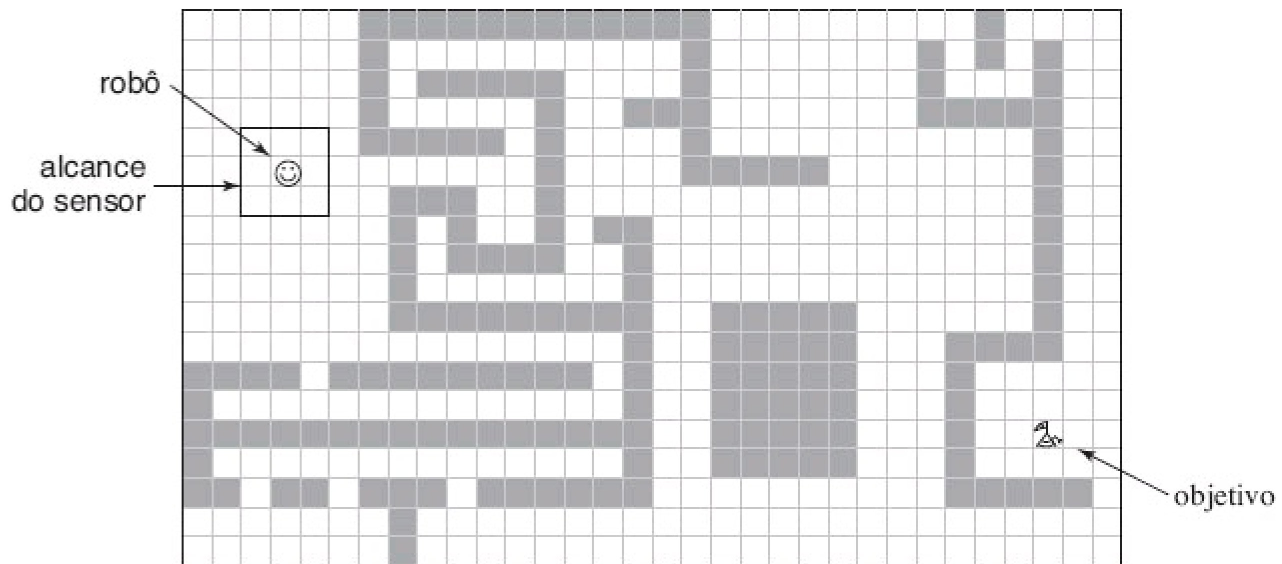


Figura 25.33 Robô simplificado em um labirinto. Veja o Exercício 25.9.

a. Projete um controlador de deliberação que garanta que o robô sempre alcançará sua posição de destino, se isso for possível. O controlador de deliberação pode memorizar medições em forma de um mapa que está sendo recebido à medida que o robô se movimenta. Entre movimentos individuais, ele pode passar um tempo arbitrário deliberando.

b. Agora, projete um controlador *reativo* para a mesma tarefa. Esse controlador não pode memorizar medições antigas de sensores (ele não pode construir um mapa!). Em vez disso, ele tem de tomar todas as decisões com base na medição atual, que inclui o conhecimento de sua própria posição e da posição do objetivo. O tempo para a tomada de uma decisão deve ser independente do tamanho do ambiente ou do número de períodos de tempo decorridos. Qual é o número máximo de etapas necessárias para que seu robô chegue ao objetivo?

c. Como seus controladores de (a) e (b) funcionarão se qualquer das seis condições a seguir se aplicar: espaço de estados contínuo, ruído na percepção, ruído no movimento, ruído tanto na percepção quanto no movimento, posição desconhecida do objetivo (o objetivo só pode ser detectado quando está dentro do alcance do sensor) ou obstáculos móveis? Para cada condição e cada controlador, dê exemplo de uma situação na qual o robô falha (ou explique por que ele não pode falhar).

25.10 Na Figura 25.24(b), encontramos uma máquina de estados finitos ampliada para o controle de uma única perna de um robô de seis pernas. Neste exercício, o objetivo é projetar primeiro uma MEFA que, quando combinada com seis cópias dos controladores de pernas individuais, resultará em locomoção eficiente e estável. Para esse fim, você tem de ampliar o controlador de perna individual para repassar mensagens à sua nova MEFA e esperar até que outras mensagens cheguem. Demonstre por que seu controlador é eficiente, no sentido de não desperdiçar energia desnecessariamente (por exemplo, deixando as pernas deslizarem) e no sentido de impelir o robô a velocidades razoavelmente altas. Prove que seu controlador satisfaz a condição de estabilidade dada na página 977.

25.11 (Este exercício foi criado primeiro por Michael Genesereth e Nils Nilsson. Ele serve para alunos desde o primeiro período até a pós-graduação.) Os seres humanos são tão hábeis em tarefas básicas como levantar xícaras ou empilhar blocos que, com frequência, se esquecem do quanto essas tarefas são complexas. Neste exercício, você descobrirá a complexidade e vai recapitular os últimos 30 anos de desenvolvimentos em robótica. Primeiro, escolha uma tarefa, como a construção de um arco usando três blocos. Em seguida, construa um robô a partir de quatro seres humanos, assim:

Cérebro. O trabalho do Cérebro é apresentar um plano para atingir o objetivo e orientar as mãos na execução do plano. O Cérebro recebe a entrada dos Olhos, mas *não pode ver a cena diretamente*. O Cérebro é o único que sabe qual é o objetivo.

Olhos. O trabalho dos Olhos é fazer uma breve descrição da cena para o Cérebro: "Existe uma caixa vermelha em posição vertical sobre uma caixa verde, que está virada de lado." Os Olhos também podem responder a perguntas do Cérebro como: "Há um intervalo entre a Mão Esquerda e a caixa vermelha?" Se tiver uma câmera de vídeo, aponte-a para a cena e deixe que os Olhos observem o visor da câmera de vídeo, mas não diretamente a cena.

Mão Esquerda e **Mão Direita**. Uma pessoa vai fazer o papel de cada Mão. As duas Mãos ficam uma ao lado da outra, cada uma vestindo uma luva de forno, em uma mão. Mãos executam apenas comandos simples do Cérebro — por exemplo, "Mão Esquerda, mova-se 5 cm à frente." Elas não podem executar comandos que não sejam movimentos; por exemplo, "Levante a caixa" não é algo que uma Mão possa fazer. As Mãos devem estar *vendadas*. A única capacidade sensória que elas devem ter é a habilidade de perceber quando seu caminho está bloqueado por um obstáculo imóvel como uma mesa ou a outra Mão. Em tais casos, elas podem emitir um aviso sonoro para informar ao Cérebro a dificuldade.

---

[1] No Capítulo 2, mencionamos **atuadores**, e não efetuadores. Um atuador é uma linha de controle que comunica um comando a um efetuador; o efetuador é o próprio dispositivo físico.

[2] "*Kinematic*" vem da palavra grega que significa *movimento*, como "cinema".

PARTE VII

Conclusão

CAPÍTULO 26

Fundamentos filosóficos

Em que consideramos o que significa pensar e se os artefatos podem e devam fazê-lo.

Os filósofos têm estado por aí há muito mais tempo que os computadores e vêm tentando resolver algumas questões que se relacionam à IA: como a mente funciona? É possível que as máquinas ajam com inteligência, de modo semelhante às pessoas, e, se isso acontecer, elas realmente terão mentes conscientes? Quais são as implicações éticas de máquinas inteligentes?

Primeiro, vejamos alguma terminologia: a asserção de que as máquinas talvez possam agir de maneira inteligente (ou, quem sabe, agir *como se* fossem inteligentes) é chamada hipótese de **IA fraca** pelos filósofos, e a asserção de que as máquinas que o fazem estão *realmente* pensando (em vez de *simularem* o pensamento) é chamada hipótese de **IA forte**.

A maior parte dos pesquisadores de IA assume em princípio a hipótese de IA fraca, e não se preocupa com a hipótese de IA forte — desde que seu programa funcione, esses pesquisadores não se importam se você o denomina simulação de inteligência ou inteligência real. Todos os pesquisadores de IA devem se preocupar com as implicações éticas de seu trabalho.

26.1 IA FRACA: AS MÁQUINAS PODEM AGIR COM INTELIGÊNCIA?

A proposta do workshop de verão de 1956, que definiu o campo da inteligência artificial (McCarthy *et al*., 1955) fez a afirmação de que "cada aspecto da aprendizagem ou qualquer outra característica da inteligência pode ser descrita tão precisamente que se pode construir uma máquina para simulá-la". Assim, estabeleceu-se IA no pressuposto de que IA fraca é possível. Outros afirmaram que IA fraca é impossível: "A inteligência artificial *procurada dentro do culto do computacionalismo* não encontra nem mesmo uma sombra de oportunidade de produzir resultados duradouros" (Sayre, 1993).

É claro que o fato de a IA ser impossível depende de como ela é definida. Na Seção 1.1, definimos IA como a busca do melhor programa de agente em dada arquitetura. Com essa formulação, a IA é possível por definição: para qualquer arquitetura digital que consiste em k bits de espaço de armazenamento, existem exatamente 2^k programas de agentes, e tudo o que temos a fazer para descobrir o melhor é enumerar e testar todos eles. Talvez isso não seja viável para k grande,

mas os filósofos lidam com o teórico, não com o prático.

Nossa definição de IA funciona bem para o problema de engenharia, que é encontrar um bom agente, dada uma arquitetura. Portanto, somos tentados a encerrar esta seção agora mesmo, respondendo afirmativamente à pergunta do título. Porém, os filósofos estão interessados no problema de comparar as duas arquiteturas — a humana e da máquina. Além disso, tradicionalmente eles formulam a questão não em termos de maximizar a utilidade esperada, mas de outra maneira: "**As máquinas podem pensar**?"

O cientista da computação Edsger Dijkstra (1984) disse que "a questão de saber se as *máquinas podem pensar* (…) é tão relevante como a questão de saber se os *submarinos podem nadar*". A primeira definição do *American Heritage Dictionary* de nadar é "mover-se através da água por meio de membros, barbatanas ou cauda", e a maioria das pessoas concorda que os submarinos não podem nadar porque não têm membros. O dicionário também define voar como "mover-se através do ar por meio de asas ou partes como asas", e a maioria das pessoas concorda que as aeronaves, tendo partes como asas, podem voar. No entanto, nem as perguntas nem as respostas têm qualquer relevância para o projeto ou a capacidade de aviões e submarinos; são sobre o uso de palavras em inglês (o fato de que os navios nadam em russo só amplifica esse ponto). A possibilidade prática de "máquinas pensantes" está conosco por apenas 50 anos ou mais, não sendo tempo suficiente para que os falantes de inglês resolvam um significado para a palavra "pensar" — requer "um cérebro" ou apenas "partes de um cérebro".

Alan Turing, em seu famoso ensaio "*Computing Machinery and Intelligence*" (Turing, 1950), sugeriu que, em vez de perguntar se as máquinas podem pensar, devemos perguntar se as máquinas podem passar por um teste de inteligência comportamental, que veio a ser chamado **teste de Turing**. O teste consiste em fazer um programa desenvolver uma conversação (via mensagens digitadas on-line) com um interrogador por cinco minutos. O interrogador deve então adivinhar se teve a conversação com um programa ou uma pessoa; o programa passa pelo teste se enganar o interrogador durante 30% do tempo. Turing conjeturou que, por volta do ano 2000, um computador com espaço de armazenamento de 10^9 unidades poderia ser programado suficientemente bem para passar no teste. Ele estava errado — os programas ainda não conseguem enganar um juiz sofisticado.

Por outro lado, muitas pessoas foram enganadas não sabendo que poderiam estar conversando com um computador. O programa ELIZA e os chatbots da Internet chamado MGONZ (Humphrys, 2008) e NATACHATA enganaram os seus correspondentes repetidamente, e o chatbot CYBERLOVER tem atraído a atenção dos policiais por causa de sua propensão para enganar com sua conversa amigável, fazendo os participantes divulgarem informações pessoais o suficiente para que sua identidade possa ser roubada. O concurso Loebner Prize, realizado anualmente desde 1991, é o concurso de execução mais longa do teste de Turing. As competições têm conduzido a modelos melhores de erros de digitação humana.

Turing também examinou ampla variedade de objeções à possibilidade de máquinas inteligentes, incluindo virtualmente todas aquelas que foram levantadas no meio século seguinte à publicação de seu trabalho. Examinaremos algumas delas.

26.1.1 O argumento de inaptidão

O “argumento de inaptidão” afirma que “uma máquina nunca poderá fazer *X*”. Como exemplos de *X*, Turing lista as seguintes tarefas:

> Ser amável, diligente, bonito, amigável, ter iniciativa, senso de humor, distinguir o certo do errado, cometer enganos, apaixonar-se, gostar de morangos e creme, fazer alguém se apaixonar por ela, aprender a partir da experiência, usar palavras corretamente, ser o sujeito de seu próprio pensamento, ter tanta diversidade de comportamento quanto o homem, fazer algo realmente novo.

Em retrospecto, algumas delas são bastante fáceis — estamos todos familiarizados com computadores que “cometem erros”. Também estamos familiarizados com uma tecnologia centenária com capacidade comprovada de “fazer alguém se apaixonar por ela” — ursos de pelúcia. O especialista em jogos de xadrez por computador, David Levy, prevê que até 2050 as pessoas rotineiramente vão se apaixonar por robôs humanoides (Levy, 2007). Paixões de robôs são um tema comum na ficção,[1] mas houve apenas especulação limitada sobre se esse fato ser provável (Kim *et al*., 2007).

Programas jogam xadrez, damas e outros jogos, inspecionam peças em linhas de montagem, dirigem carros e helicópteros, verificam a grafia de documentos de processamento de textos, dirigem automóveis e pilotam helicópteros, fazem o diagnóstico de doenças e executam centenas de outras tarefas, tão bem ou melhor que os seres humanos. Os computadores fizeram descobertas pequenas mas significativas em astronomia, matemática, química, mineralogia, biologia, ciência da computação e outros campos. Cada uma dessas descobertas exigiu desempenho no nível de um especialista humano.

Dado o que sabemos agora sobre computadores, não surpreende que eles se comportem tão bem em problemas combinatórios como jogar xadrez. Porém, os algoritmos também funcionam em níveis humanos em tarefas que aparentemente envolvem julgamento humano ou, como Turing observou, “aprender a partir da experiência” e a capacidade de “distinguir o certo do errado”. Desde 1955, Paul Meehl (veja também Grove e Meehl, 1996) estudou os processos de tomada de decisão de especialistas treinados em tarefas subjetivas como prever o sucesso de um aluno em um programa de treinamento ou a reincidência de um criminoso. Em 19 dos 20 estudos que examinou, Meehl descobriu que algoritmos simples de aprendizado estatístico (como regressão linear ou Bayes ingênuo) fizeram previsões melhores que os especialistas. O Educational Testing Service utilizou um programa automatizado para avaliar milhões de perguntas de composição no exame de GMAT desde 1999. O programa concorda com avaliadores humanos durante 97% do tempo, aproximadamente o mesmo nível de concordância entre dois avaliadores humanos (Burstein *et al.,* 2001).

É claro que os computadores podem fazer muitas coisas tão bem ou melhor que os humanos, incluindo aquelas que as pessoas acreditam que exigem grande perspicácia e compreensão humana. No entanto, isso não significa que os computadores utilizam a perspicácia e a compreensão na execução dessas tarefas — essas características não fazem parte do *comportamento*, e examinaremos tais questões em outro lugar —, mas o importante é que a primeira suposição de alguém sobre os processos mentais exigidos para produzir dado comportamento com frequência está errada. Evidentemente, também é verdade que existem muitas tarefas em que os computadores não têm desempenho excelente (para sermos benevolentes), incluindo a tarefa de Turing de desenvolver uma conversação ilimitada.

26.1.2 A objeção matemática

Sabemos muito bem, graças ao trabalho de Turing (1936) e Gödel (1931), que certas questões matemáticas são em princípio insolúveis para sistemas formais específicos. O teorema da incompleteza de Gödel (veja a Seção 9.5) é o exemplo mais famoso desse fato. Em resumo, para qualquer sistema axiomático formal F poderoso o suficiente para efetuar operações aritméticas, é possível construir uma "sentença de Gödel" $G(F)$ com as seguintes propriedades:

- $G(F)$ é uma sentença de F, mas não pode ser provada dentro de F.
- Se F é consistente, então $G(F)$ é verdadeira.

Filósofos como J. R. Lucas (1961) afirmam que esse teorema mostra que as máquinas são mentalmente inferiores aos seres humanos porque as máquinas são sistemas formais limitados pelo teorema da incompleteza — não podem estabelecer a verdade sobre sua própria sentença de Gödel —, enquanto os seres humanos não têm tal limitação. Essa afirmativa gerou décadas de controvérsia, produzindo vasta literatura que incluiu dois livros escritos pelo matemático *Sir* Roger Penrose (1989, 1994) que repetiu essa afirmativa com algumas variações novas (como a hipótese de que os seres humanos são diferentes porque seu cérebro opera por gravidade quântica). Examinaremos apenas três dos problemas relacionados com essa afirmativa.

Primeiro, o teorema da incompleteza de Gödel se aplica apenas a sistemas formais poderosos o suficiente para realizar cálculos aritméticos. Isso inclui as máquinas de Turing, e a afirmativa de Lucas se baseia em parte na asserção de que os computadores são máquinas de Turing. Essa é uma boa aproximação, mas não é muito verdadeira. As máquinas de Turing são infinitas, enquanto os computadores são finitos, e qualquer computador pode então ser descrito como um sistema (muito grande) em lógica proposicional que não está sujeito ao teorema da incompleteza de Gödel.

Em segundo lugar, um agente não deve ficar muito envergonhado se não puder estabelecer a verdade de alguma sentença, enquanto outros agentes conseguem fazê-lo. Considere a sentença:

J. R. Lucas não pode afirmar de forma consistente que essa sentença é verdadeira.

Se Lucas afirmasse essa sentença, ele estaria contradizendo a si mesmo, e portanto Lucas não pode afirmá-la de forma consistente; consequentemente, ela tem de ser verdadeira. Desse modo, demonstramos que existe uma sentença que Lucas não pode afirmar de forma consistente, enquanto outras pessoas (e máquinas) podem. No entanto, isso não diminui nossa consideração por Lucas. Como outro exemplo, nenhum ser humano seria capaz de calcular a soma de 10 bilhões de números de 10 dígitos durante seu tempo de vida, mas um computador poderia fazê-lo em segundos. Ainda assim, não vemos esse fato como limitação fundamental da habilidade humana para pensar. Os seres humanos se comportaram de forma inteligente durante milhares de anos antes de criarem a matemática; assim, é improvável que o raciocínio matemático desempenhe um papel mais do que periférico no significado de ser inteligente.

Em terceiro lugar, e mais importante, ainda que aceitemos que os computadores têm limitações sobre o que podem provar, não existe nenhuma evidência de que os seres humanos sejam imunes a

essas limitações. É muito fácil mostrar de maneira rigorosa que um sistema formal não pode realizar *X* e, em seguida, afirmar que os seres humanos *podem* realizar *X* usando seu próprio método informal sem dar qualquer evidência em apoio a essa afirmação. Na realidade, é impossível provar que os seres humanos não estão sujeitos ao teorema da incompleteza de Gödel porque qualquer prova rigorosa conteria ela própria uma formalização do talento humano considerado informalizável e, consequentemente, seria uma refutação de si mesma. Então, temos de apelar para a intuição de que os seres humanos podem de alguma forma executar feitos super-humanos de habilidade matemática. Esse apelo é expresso por argumentos como "devemos supor nossa própria coerência se o pensamento de fato deve ser possível" (Lucas, 1976). Porém, os seres humanos são reconhecidamente incoerentes. Sem dúvida, isso é verdadeiro no caso do raciocínio do dia a dia, mas também é verdadeiro para o pensamento matemático cuidadoso. Um exemplo famoso é o problema do mapa de quatro cores. Alfred Kempe publicou uma prova em 1879 extensamente aceita e que colaborou para sua eleição como membro do conselho da Royal Society. Porém, em 1890, Percy Heawood assinalou uma falha e o teorema permaneceu sem demonstração até 1977.

26.1.3 O argumento da informalidade

Uma das críticas mais influentes e persistentes à IA como empreendimento foi levantada por Turing como o "argumento da informalidade do comportamento". Em essência, essa é a afirmação de que o comportamento humano é complexo demais para ser capturado por qualquer conjunto simples de regras e que, como os computadores não podem fazer nada mais além de seguir um conjunto de regras, eles não podem gerar comportamento tão inteligente quanto o dos seres humanos. A inabilidade para reunir tudo em um conjunto de regras lógicas é chamado **problema de qualificação** em IA.

O principal proponente dessa visão foi o filósofo Hubert Dreyfus, que produziu uma série de críticas influentes para a inteligência artificial: *What Computers Can't Do* (1972), *What Computers Still Can't Do* (1992) e, com seu irmão Stuart, *Mind Over Machine* (1986).

A posição que eles criticaram veio a ser chamada "Good Old-Fashioned AI" ("a boa e velha IA"), ou GOFAI, um termo cunhado pelo filósofo John Haugeland (1985). A GOFAI afirma que todo comportamento inteligente pode ser captado por um sistema que raciocine logicamente a partir de um conjunto de fatos e regras que descrevem o domínio. Portanto, corresponde ao agente lógico mais simples descrito no Capítulo 7. Dreyfus está correto ao afirmar que agentes lógicos são vulneráveis ao problema de qualificação. Como vimos no Capítulo 13, sistemas de raciocínio probabilístico são mais apropriados para domínios ilimitados. A crítica de Dreyfus não é dirigida aos computadores em si, mas a uma forma específica de programá-los. Entretanto, é razoável supor que um livro intitulado *What First-Order Logical Rule-Based Systems Without Learning Can't Do* ("O que sistemas baseados em regras de lógica de primeira ordem sem aprendizagem não podem fazer") talvez causasse menor impacto.

Sob a visão de Dreyfus, a perícia humana inclui o conhecimento de algumas regras, mas apenas como "contexto holístico" ou "pano de fundo" dentro do qual os seres humanos operam. Ele fornece o exemplo de comportamento social apropriado ao dar e receber presentes: "Em geral, em

circunstâncias apropriadas, nossa resposta é simplesmente dar um presente adequado.” Aparentemente, temos um “sentido direto de como as coisas são feitas e do que devemos esperar”. A mesma afirmação é feita no contexto do jogo de xadrez: “Um simples mestre de xadrez talvez precisasse compreender o que fazer, mas um grande mestre só vê o tabuleiro como a exigência de certo movimento (…) a resposta correta simplesmente salta de seu cérebro.” Sem dúvida, é verdade que grande parte dos processos de pensamento de quem oferece um presente ou de um grande mestre de xadrez é realizada em nível que não está aberto à introspecção pela mente consciente. Porém, isso não significa que os processos de pensamento não existam. A importante questão que Dreyfus não responde é *como* o movimento correto entra na cabeça do grande mestre. Lembramos aqui um comentário de Daniel Dennett (1984):

> É como se os filósofos se proclamassem especialistas na explicação dos métodos utilizados pelos mágicos de palco e, depois, quando perguntamos como o mágico realiza o truque de serrar sua assistente ao meio, eles explicam que de fato é óbvio: na verdade, o mágico não serra a assistente ao meio; ele simplesmente faz parecer que a serrou ao meio. “Mas *como* ele faz isso?”, perguntamos. “Não é nosso departamento”, dizem os filósofos.

Dreyfus e Dreyfus (1986) propõem um processo de aquisição de experiência em cinco fases, começando com o processamento baseado em regras (do tipo proposto na GOFAI) e terminando com a habilidade para selecionar instantaneamente respostas corretas. Ao fazer essa proposta, Dreyfus e Dreyfus passam de críticos da IA a teóricos da IA — eles propõem uma arquitetura de rede neural organizada em uma vasta “biblioteca de casos”, mas assinalam diversos problemas. Felizmente, todos os seus problemas foram tratados, alguns com sucesso parcial e outros com sucesso total. Seus problemas incluem:

1. Boa generalização a partir de exemplos não pode ser alcançada sem conhecimento prático. Eles afirmam que ninguém tem qualquer ideia de como incorporar o conhecimento prático ao processo de aprendizado de redes neurais. De fato, vimos nos Capítulos 19 e 20 que existem técnicas para usar o conhecimento anterior em algoritmos de aprendizado. Porém, essas técnicas se baseiam na disponibilidade de conhecimento em forma explícita, algo que Dreyfus e Dreyfus negam enfaticamente. Em nossa visão, essa é uma boa razão para um séria redefinição dos modelos atuais de processamento neural, para que eles *possam* tirar proveito do conhecimento aprendido anteriormente, como fazem outros algoritmos de aprendizado.
2. O aprendizado de redes neurais é uma forma de aprendizado supervisionado (veja o Capítulo 18), exigindo a identificação anterior de entradas relevantes e saídas corretas. Portanto, eles afirmam que esse aprendizado não pode operar de modo autônomo sem a ajuda de um treinador humano. De fato, o aprendizado sem professor pode ser realizado por **aprendizado não supervisionado** (Capítulo 20) e **aprendizado por reforço** (Capítulo 21).
3. Os algoritmos de aprendizado não funcionam bem com muitas características e, se escolhermos um subconjunto de características, “não haverá nenhum modo conhecido de adicionar novas características, caso o conjunto atual se mostre inadequado para levar em conta os fatos aprendidos”. De fato, novos métodos como máquinas de vetor de suporte manipulam muito bem grandes conjuntos de características. Com a introdução dos amplos conjuntos de dados

baseados na Web, muitos aplicativos em áreas como a de processamento de linguagem (Sha e Pereira, 2003) e visão de computador (Viola e Jones, 2002a) manuseiam milhões de características. Como vimos no Capítulo 19, também existem meios consistentes para gerar novas características, embora seja necessário muito mais trabalho.

4. O cérebro é capaz de orientar seus sensores para buscar informações relevantes e processá-las com o objetivo de extrair aspectos relevantes para a situação atual. No entanto, Dreyfus e Dreyfus afirmam que "atualmente, nenhum detalhe desse mecanismo é compreendido ou mesmo visto como hipótese de modo que possa orientar a pesquisa em IA". De fato, o campo da visão ativa, admitido pela teoria de valor da informação (Capítulo 16), se preocupa exatamente com o problema de orientar sensores, e alguns robôs já incorporaram os resultados teóricos obtidos. A viagem de STANLEY de 132 milhas através do deserto foi possível, em grande parte, por um sistema de sensor ativo desse tipo.

Em suma, muitas das questões em que Dreyfus se concentrou — conhecimento prático de senso comum, problema de qualificação, incerteza, aprendizado, formas compiladas de tomada de decisões, importância de considerar agentes situados em vez de mecanismos de inferência dispersos — estão agora incorporadas ao projeto de agentes inteligentes-padrão. Em nossa visão, essa é a evidência do progresso da IA, não de sua impossibilidade.

Um dos argumentos mais fortes de Dreyfus é dos agentes situados em vez de mecanismos de inferência lógica desprovidos de realização física. Um agente cuja compreensão de "cachorro" só vem de um conjunto limitado de sentenças lógicas, como "$Cão(x) \Rightarrow Mamífero(x)$" está em desvantagem em comparação com um agente que viu cachorros correr, brincou de persegui-los e foi lambido por um. Como o filósofo Andy Clark (1998) diz: "Cérebros biológicos são em primeiro lugar o sistemas de controle de organismos biológicos. Corpos biológicos movem-se e agem no ambiente do rico mundo real."

Para entender como os agentes humanos (ou animais) funcionam, temos de considerar o agente como um todo, não apenas o programa do agente. Na verdade, a abordagem de **cognição incorporada** alega que não faz sentido considerar o cérebro separadamente: a cognição ocorre dentro do corpo, que está incorporado em um ambiente. Precisamos estudar o sistema como um todo; o cérebro aumenta seu raciocínio quando se refere ao ambiente, como o leitor ao perceber (e criar) marcas no papel para transferência de conhecimento. No âmbito do programa de cognição incorporada, robótica, visão e outros sensores tornam-se centrais, não periféricos.

26.2 IA FORTE: AS MÁQUINAS PODEM REALMENTE PENSAR?

Muitos filósofos afirmam que uma máquina que passasse pelo teste de Turing ainda não estaria *realmente* pensando, mas seria apenas uma *simulação* de pensamento. Mais uma vez, a objeção foi prevista por Turing. Ele cita uma conferência do professor Geoffrey Jefferson (1949):

> Somente quando uma máquina conseguir escrever um soneto ou compor um concerto em consequência de ter pensado e sentido emoções, e não pela disposição aleatória de símbolos,

poderemos concordar que a máquina vai se equiparar ao cérebro, isto é, se ela não apenas escrever, mas souber que escreveu.

Turing chama esse argumento de **consciência** — a máquina tem de estar ciente de seus próprios estados mentais e de suas ações. Embora a consciência seja um assunto importante, o ponto fundamental destacado por Jefferson se relaciona na verdade à **fenomenologia**, ou o estudo da experiência direta — a máquina precisa realmente sentir emoções. Outros se concentram na **intencionalidade**, ou seja, na questão de saber se as crenças, os desejos e outras representações da máquina são realmente "sobre" algo que existe no mundo real.

A resposta de Turing à objeção é interessante. Ele poderia ter apresentado razões para explicar que as máquinas podem de fato ser conscientes (ou ter fenomenologia, ou ter intenções). Em vez disso, ele sustenta que a questão está tão mal definida quanto a pergunta: "As máquinas podem pensar?" Além disso, por que devemos insistir em um padrão mais alto para as máquinas do que adotamos para os seres humanos? Afinal, na vida comum, nunca temos *qualquer* evidência direta sobre os estados mentais internos de outros seres humanos. Não obstante, Turing diz: "Em vez de discutir continuamente sobre esse ponto, é habitual assumir a convenção cortês de que todos pensam."

Turing argumenta que Jefferson estaria disposto a estender a convenção cortês para máquinas somente se tivesse experiência com máquinas que agissem de modo inteligente. Ele cita o seguinte diálogo, que se tornou uma parte tão importante da tradição oral da IA que simplesmente temos de incluí-lo:

HUMANO: na primeira linha do soneto lê-se "devo te comparar a um dia de sol," um "dia de primavera" daria no mesmo ou ficaria melhor?
MÁQUINA: não faria a busca.
HUMANO: e um "dia de inverno." Nesse caso faria a busca.
MÁQUINA: sim, mas ninguém quer ser comparado a um dia de inverno.
HUMANO: você diria que o Sr. Pickwick lembrou-o do Natal?
MÁQUINA: de certo modo.
HUMANO: o Natal é em um dia de inverno e não acredito que o Sr. Pickwick iria se importar com a comparação.
MÁQUINA: acho que você não está falando sério. Pois um dia de inverno significa um dia de inverno típico, não um dia especial como o Natal.

Pode-se facilmente imaginar algum tempo futuro em que essas conversas com as máquinas serão comuns, e se tornará costume não fazer nenhuma distinção linguística entre pensamento "real" e "artificial". Uma transição semelhante ocorreu nos anos após 1848, quando a ureia artificial foi sintetizada pela primeira vez por Frederick Wöhler. Antes desse evento, a química orgânica e a química inorgânica eram essencialmente empreendimentos disjuntos e muitos pensavam que não poderia existir nenhum processo que convertesse compostos químicos inorgânicos em matéria orgânica. Uma vez realizada a síntese, os químicos concordaram que a ureia artificial era ureia porque tinha todas as propriedades físicas corretas. Os que postularam uma propriedade intrínseca presente nos materiais orgânicos pela qual eles nunca pudessem ser preparados a partir de materiais

orgânicos depararam-se com a impossibilidade de elaboração de qualquer teste que pudesse revelar a suposta deficiência da ureia artificial.

Com relação ao pensamento ainda não atingimos nosso 1848, e há aqueles que acreditem que o pensamento artificial, não importa o quão impressionante, nunca será real. Por exemplo, o filósofo John Searle (1980) argumenta que:

> Ninguém supõe que uma simulação de computador de uma tempestade nos deixará molhados (…) então, por que alguém em seu juízo perfeito iria supor que uma simulação de computador de processos mentais realmente teria processos mentais? (p. 37-38)

Embora seja fácil concordar com o fato de que simulações de tempestades por computador não nos molham, não está claro como transportar essa analogia para simulações por computador de processos mentais. Afinal, uma simulação feita por Hollywood de uma tempestade usando irrigadores e máquinas de vento *deixa* os atores molhados. A maioria das pessoas se sente confortável afirmando que a simulação da adição por um computador é uma adição e que a simulação por computador de um jogo de xadrez é um jogo de xadrez. Na verdade, geralmente falamos em *implementação* de resultado ou de xadrez, e não de *simulação.* Os processos mentais são mais tempestades ou mais adição?

A resposta de Turing — a convenção cortês — sugere que a questão acabará se desvanecendo por si só, uma vez que as máquinas alcancem certo nível de sofisticação. Isso terá o efeito de *dissolver* a diferença entre IA fraca e forte. Contra isso pode-se insistir que há uma questão *factual* em jogo: os seres humanos têm mentes reais, e as máquinas podem ter ou não. Para abordar essa questão factual, precisamos entender como os seres humanos têm mentes reais, não apenas corpos que geram processos neurofisiológicos. Os esforços filosóficos para resolver esse **problema mente-corpo** são diretamente relevantes para a questão de que as máquinas podem ter mentes reais.

O problema mente-corpo foi considerado pelos filósofos da Grécia antiga e por várias escolas de pensamento hindu, mas foi analisado em profundidade primeiramente pelo filósofo e matemático francês do século XVII René Descartes. Seu livro *Meditações sobre a filosofia primeira* (1641) considerava a atividade da mente de pensar (um processo sem extensão espacial ou propriedades materiais) e os processos físicos do corpo, concluindo que os dois deviam existir em reinos separados — o que hoje chamaríamos de teoria **dualista**. O problema mente-corpo enfrentado pelos dualistas é a questão de como a mente pode controlar o corpo, se os dois estão realmente separados. Descartes especulou que os dois podem interagir através da glândula pineal, que simplesmente transfere o problema para a questão de como a mente controla a glândula pineal.

A teoria **monista** da mente, muitas vezes chamada de **fisicalismo**, evita esse problema afirmando que a mente não é separada do corpo — os estados mentais *são* estados físicos. Os filósofos da mente mais modernos são fisicalistas de uma forma ou de outra, e o fisicalismo permite, pelo menos em princípio, a possibilidade da IA forte. O problema para os fisicalistas é explicar como estados físicos — em particular, as configurações moleculares e os processos eletroquímicos do cérebro — podem ser simultaneamente **estados mentais**, como dor, saborear um hambúrguer, saber que se está montando em um cavalo ou acreditar que Viena é a capital da Áustria.

26.2.1 Os estados mentais e o cérebro numa cuba

Os filósofos fisicalistas tentaram explicar o que significa dizer que uma pessoa — e, por extensão, um computador — está em determinado estado mental. Eles se concentraram em especial sobre **estados intencionais**. São estados como acreditar, conhecer, desejar, temer, e assim por diante, que se referem a algum aspecto do mundo externo. Por exemplo, o conhecimento de estar comendo um hambúrguer é uma crença *sobre* o hambúrguer e o que está acontecendo com ele.

Se o fisicalismo estiver correto, deve ser o caso que a descrição adequada de um estado mental de uma pessoa seja *determinada* pelo estado do cérebro da pessoa. Assim, se eu estou atualmente focado em comer um hambúrguer de forma atenta, meu estado cerebral instantâneo é uma instância da classe de estados mentais "saber que se está comendo um hambúrguer". É claro que as configurações específicas de todos os átomos do meu cérebro não são essenciais: há muitas configurações do meu cérebro, ou do cérebro de outras pessoas, que pertencem à mesma classe de estados mentais. O ponto-chave é que o mesmo estado do cérebro não poderia corresponder a um estado mental fundamentalmente distinto, tal como o conhecimento de que se está comendo uma banana.

A simplicidade dessa visão é contestada por alguns experimentos mentais simples. Imagine se você desejasse que seu cérebro fosse removido de seu corpo no momento do nascimento e colocado em uma cuba maravilhosamente montada. A cuba sustentará o seu cérebro, permitindo que cresça e se desenvolva. Ao mesmo tempo, sinais eletrônicos de um mundo inteiramente fictício são enviados para o seu cérebro a partir de uma simulação em computador, e os sinais motrizes do seu cérebro são interceptados e utilizados para modificar a simulação conforme apropriado.[2] Na verdade, a vida simulada que você vive reproduz exatamente a vida que você teria vivido se o seu cérebro não tivesse sido colocado na cuba, incluindo alimentação simulada com hambúrgueres simulados. Assim, você poderia ter um estado cerebral idêntico ao de alguém que está realmente comendo um hambúrguer de verdade, mas seria literalmente falso dizer que você tem o estado mental "saber que está comendo um hambúrguer". Você não está comendo um hambúrguer, nunca sequer experimentou um hambúrguer e não poderia, portanto, ter esse estado mental.

Esse exemplo parece contradizer a visão de que os estados cerebrais determinam estados mentais. Uma maneira de resolver o dilema é dizer que o conteúdo dos estados mentais pode ser interpretado a partir de dois pontos de vista diferentes. A visão do "**conteúdo amplo**" interpreta do ponto de vista de um observador onisciente externo com acesso a toda a situação, que pode distinguir diferenças no mundo. Sob essa ótica, o conteúdo dos estados mentais envolve tanto o estado do cérebro como a história do ambiente. O **conteúdo restrito**, por outro lado, considera apenas o estado do cérebro. O conteúdo restrito dos estados cerebrais de um verdadeiro comedor de hambúrguer e de um "comedor" de "hambúrguer" de um cérebro em uma cuba é o mesmo em ambos os casos.

O conteúdo amplo é totalmente apropriado se o objetivo da pessoa for o de atribuir estados mentais aos outros que partilham o seu mundo, para prever seu comportamento provável e seus efeitos, e assim por diante. Esse é o cenário em que a nossa linguagem comum sobre o conteúdo mental evoluiu. Por outro lado, se alguém está preocupado com a questão de os sistemas de IA serem realmente pensantes e realmente terem estados mentais, o conteúdo restrito é apropriado; simplesmente não faz sentido dizer que o fato de um sistema de inteligência artificial ser realmente pensante ou não depende das condições externas desse sistema. O conteúdo restrito também é

relevante se estamos pensando em projetar sistemas de IA ou compreender o seu funcionamento, pois é o conteúdo restrito de um estado do cérebro que determina qual será o (conteúdo restrito do) próximo estado cerebral. Isso leva naturalmente à ideia de que o que importa sobre um estado cerebral — o que faz com que ele tenha um tipo de conteúdo mental e não outro — é o seu papel funcional dentro da operação mental da entidade envolvida.

26.2.2 O funcionalismo e o experimento de prótese cerebral

A teoria do **funcionalismo** diz que um estado mental é qualquer condição causal intermediária entre a entrada e a saída. De acordo com a teoria funcionalista, quaisquer dois sistemas com processos causais isomórficos teriam os mesmos estados mentais. Portanto, um programa de computador poderia ter os mesmos estados mentais que uma pessoa. Claro, ainda não dissemos o que "isomórfico" significa realmente, mas a suposição é que há algum nível de abstração abaixo do qual a implementação específica não importa.

As reivindicações do funcionalismo são ilustradas mais claramente pelo experimento de prótese cerebral. Esse experimento de pensamento foi introduzido pelo filósofo Clark Glymour e foi abordado por John Searle (1980), mas é mais comumente associado com o roboticista Hans Moravec (1988). É assim: suponha que a neurofisiologia desenvolveu-se ao ponto em que o comportamento de entrada e saída e a conectividade de todos os neurônios do cérebro humano são perfeitamente compreendidos. Além disso suponha que possamos construir dispositivos eletrônicos microscópicos que imitem esse comportamento e que possam ser facilmente incorporados ao tecido neural. Por fim, suponha que, por algum milagre, a técnica cirúrgica possa substituir os neurônios individuais por dispositivos eletrônicos correspondentes sem interromper o funcionamento do cérebro como um todo. O experimento consiste em substituir gradualmente todos os neurônios na cabeça de alguém com dispositivos eletrônicos.

Estamos preocupados tanto com o comportamento externo como com a experiência interna do indivíduo, durante e após a operação. Pela definição do experimento, o comportamento externo do indivíduo deve permanecer inalterado em comparação com o que seria observado se a operação não fosse efetuada.[3] Agora, embora a presença ou ausência de consciência não possa ser verificada facilmente por um terceiro, o indivíduo da experiência deveria pelo menos ser capaz de gravar algumas mudanças em sua própria experiência consciente. Aparentemente, há um confronto direto de intuições sobre o que iria acontecer. Moravec, pesquisador de robótica e funcionalista, está convencido de que sua consciência permaneceria inalterada. Searle, filósofo e biólogo naturalista, está convencido igualmente de que sua consciência desapareceria:

> Para espanto total, você pensa que está realmente perdendo o controle do seu comportamento externo. Pensa, por exemplo, que, quando os médicos testam a sua visão, você os ouve dizer: "Estamos segurando um objeto vermelho na sua frente, por favor, diga-nos o que você vê." Você quer gritar: "Eu não posso ver nada. Estou ficando totalmente cego." Mas ouve sua voz dizendo que você está completamente fora de controle: "Eu vejo um objeto vermelho na minha frente." (…) sua experiência consciente afunda lentamente para o nada, enquanto o seu comportamento observável externo permanece o mesmo. (Searle, 1992)

Pode-se fazer mais do que argumentar a partir da intuição. Primeiro, observe que, para que o comportamento externo permaneça o mesmo enquanto o indivíduo for ficando gradualmente inconsciente, é preciso que a vontade do indivíduo seja removida instantânea e totalmente; caso contrário, a diminuição da consciência seria refletida no comportamento externo — "Socorro, estou perdendo a consciência!" ou palavras que surtam o mesmo efeito. Essa remoção instantânea da volição como resultado da substituição gradual de um neurônio por vez parece uma alegação pouco provável.

Segundo, considere o que acontece se fizermos perguntas ao indivíduo a respeito de sua experiência consciente durante o período em que nenhum neurônio real permanece. Pelas condições do experimento, vamos obter respostas como: "Eu me sinto bem. Devo dizer que estou um pouco surpreso porque acreditava no argumento de Searle." Ou podemos picar o indivíduo com um espeto pontiagudo e observar a resposta: "Ai, que dor." Agora, no curso normal dos acontecimentos, o cético pode descartar tais saídas dos programas de IA como simples invenções. Certamente, é suficientemente fácil utilizar uma regra tal como "Se sensor 12 lê 'Alto', então informe 'Ai'." Mas o ponto aqui é que, como replicamos as propriedades funcionais de um cérebro humano normal, assumimos que o cérebro eletrônico não contém tais artifícios. Então temos de ter uma explicação de manifestações da consciência produzida pelo cérebro eletrônico que apela apenas para as propriedades funcionais dos neurônios. *E essa explicação deve também se aplicar ao cérebro real, que tem as mesmas propriedades funcionais.* Há três possíveis conclusões:

1. Os mecanismos causais da consciência que geram esses tipos de saídas em cérebros normais ainda estão operando na versão eletrônica, que é, portanto, consciente.
2. Os eventos mentais conscientes no cérebro normal não têm nenhuma conexão causal com o comportamento e estão ausentes no cérebro eletrônico, que é, portanto, não consciente.
3. O experimento é impossível e, portanto, a especulação sobre ele não tem sentido.

Embora não possamos descartar a segunda possibilidade, ela reduz a consciência para o que os filósofos chamam de papel de **epifenômeno** — algo que acontece, sem lançar sombra, por assim dizer, sobre o mundo observável. Além disso, se a consciência for de fato um epifenômeno, pode ser que a cobaia não diga "Ai" *porque dói* — isto é, por causa da experiência consciente da dor. Em vez disso, o cérebro deve conter um segundo mecanismo, inconsciente, responsável pelo "Ai".

Patricia Churchland (1986) mostra que os argumentos funcionalistas que operam em nível de neurônio também podem operar em nível de qualquer unidade maior — um grupo de neurônios, um módulo mental, um lóbulo, um hemisfério ou o cérebro todo. Isso significa que, se você aceita a noção de que o experimento de prótese cerebral mostra que a o cérebro substituído está consciente, você também deve acreditar que a consciência será mantida quando todo o cérebro for substituído por um circuito que atualiza o seu estado e faz o mapeamento das entradas para as saídas por meio de uma enorme tabela de pesquisa. Isso é desconcertante para muitas pessoas (incluindo Turing) que têm a intuição de que as tabelas de pesquisa não são conscientes — ou, pelo menos, que as experiências conscientes geradas durante a pesquisa à tabela não são as mesmas que as geradas durante a operação de um sistema que pode ser descrito (mesmo em sentido computacional simplório) como para acessar e gerar crenças, introspecções, objetivos, e assim por diante.

26.2.3 Naturalismo biológico e o quarto chinês

Um forte desafio para o funcionalismo foi criado pelo **naturalismo biológico** de John Searle (1980); de acordo com ele, os estados mentais são características emergentes de alto nível causadas pelo baixo nível de processos físicos *nos neurônios* e as propriedades (não especificadas) dos neurônios é que são importantes. Assim, os estados mentais não podem ser duplicados apenas por algum programa ter a mesma estrutura funcional com o mesmo comportamento de entrada e saída; exigiríamos que o programa fosse executado em uma arquitetura com o mesmo poder causal que os neurônios. Para apoiar essa visão, Searle descreve um sistema hipotético que de fato está executando um programa e passa pelo teste de Turing, mas que de forma igualmente clara (de acordo com Searle) não *entende* nada de suas entradas e saídas. Sua conclusão é de que a execução do programa apropriado (isto é, ter as saídas corretas) não é uma condição *suficiente* para ser uma mente.

O sistema consiste em um ser humano, que compreende apenas o idioma inglês, equipado com um livro de regras escrito em inglês e diversas pilhas de papel, sendo algumas em branco e algumas com inscrições indecifráveis (portanto, o ser humano faz o papel da CPU, o livro de regras é o programa, e as pilhas de papel são o dispositivo de armazenamento). O sistema está em um quarto com uma pequena abertura para o exterior. Por essa abertura passam folhas de papel com símbolos indecifráveis. O ser humano encontra símbolos correspondentes no livro de regras e segue as instruções. As instruções podem incluir escrever símbolos em novas folhas de papel, encontrar símbolos nas pilhas, reorganizar as pilhas, e assim por diante. Eventualmente, as instruções farão com que um ou mais símbolos sejam transcritos em uma folha de papel que será repassada de volta ao mundo exterior.

Até agora, tudo bem. Porém, do exterior, observamos um sistema que está recebendo a entrada sob a forma de sentenças em chinês e gerando respostas em chinês que são sem dúvida "inteligentes", como aquelas da conversação imaginada por Turing.[4] Então, Searle argumenta da seguinte forma: a pessoa no quarto não compreende chinês (dado). O livro de regras e as pilhas de papel, sendo apenas folhas de papel, não entendem chinês. Então, não está acontecendo nenhuma compreensão de chinês. *Por conseguinte, de acordo com Searle, a execução do programa correto não gera necessariamente a compreensão*.

Como Turing, Searle considerou e tentou repelir várias respostas ao seu argumento. Diversos comentaristas, inclusive John McCarthy e Robert Wilensky, propuseram aquilo que Searle chama de resposta de sistemas. A objeção é que, embora se possa perguntar se o ser humano no quarto entende chinês, isso é análogo a perguntar se a CPU pode efetuar raízes cúbicas. Em ambos os casos, a resposta é não, e, em ambos os casos, de acordo com a resposta de sistemas, o sistema inteiro *tem* a capacidade em questão. Certamente, se alguém perguntasse ao quarto chinês se ele compreende chinês, a resposta seria afirmativa (em chinês fluente). Pela convenção cortês de Turing, isso deve ser suficiente. A resposta de Searle é reiterar a afirmação de que a compreensão não está no ser humano e não pode estar no papel; assim, não pode haver qualquer compreensão. Ele parece estar contando com o argumento de que uma propriedade do todo deve residir em uma das partes.

A água é molhada, mesmo que H e O_2 não o sejam. A afirmação real feita por Searle serve de base aos quatro axiomas a seguir (Searle, 1990):

1. Os programas de computador são formais (entidades sintáticas).
2. As mentes humanas têm conteúdo mental ou semântico.
3. Sozinha, a sintaxe não é constitutiva nem suficiente para a semântica.
4. Cérebros causam mentes.

A partir dos três primeiros axiomas, ele conclui que os programas não são suficientes para mentes. Em outras palavras, um agente que executa um programa pode não ser uma mente, mas não precisa ser uma mente apenas pelo fato de executar o programa. A partir do quarto axioma, ele conclui que "qualquer outro sistema capaz de causar mentes teria de apresentar poderes causais (pelo menos) equivalentes aos do cérebro". A partir daí, ele deduz que qualquer cérebro artificial teria de reproduzir os poderes causais do cérebro, não apenas executar um programa específico, e que os cérebros humanos não produzem fenômenos mentais somente em virtude da execução de um programa.

Os axiomas são controversos. Por exemplo, os axiomas 1 e 2 contam com uma distinção não especificada entre sintaxe e semântica que parece estar intimamente relacionada com a distinção entre o conteúdo restrito e amplo. Por um lado, podemos ver os computadores como manipuladores de símbolos sintáticos; por outro lado, podemos vê-los como manipuladores de corrente elétrica, que acontece de ser o que o cérebro faz na maior parte (de acordo com nossa compreensão atual). Assim, parece que igualmente poderíamos dizer que os cérebros são sintáticos.

Assumindo que somos generosos na interpretação dos axiomas, a conclusão de que os programas não são suficientes para mentes procede. Mas a conclusão não é satisfatória — tudo o que Searle tem mostrado é que, se negar explicitamente o funcionalismo (que é o que o seu axioma 3 faz), você não pode concluir necessariamente que não cérebros sejam mentes. Isso é bastante razoável — quase tautológico; assim, todo o argumento se resume em saber se o axioma 3 pode ser aceito. De acordo com Searle, o ponto do argumento do quarto chinês é fornecer intuições para o axioma 3. A reação pública mostra que o argumento está agindo como o que Daniel Dennett (1991) chama de **bomba de intuição**: ela amplifica as intuições anteriores, os naturalistas biológicos ficam mais convencidos de suas posições e os funcionalistas ficam convencidos apenas de que o axioma 3 não se sustenta ou, em geral, que o argumento de Searle não é convincente. O argumento incitou embates, mas fez pouco para mudar a opinião de alguém. Searle permanece irredutível e, recentemente, começou a chamar o quarto chinês de "refutação" de IA forte, em vez de apenas "argumento" (Snell, 2008).

Mesmo aqueles que aceitam o axioma 3, e portanto aceitam o argumento de Searle, só podem recorrer a suas intuições ao decidir que entidades são mentes. O argumento pretende mostrar que o quarto chinês não é uma mente *em virtude da execução do programa*, mas o argumento não diz nada sobre como decidir se o quarto (ou um computador, algum outro tipo de máquina ou um alienígena) é uma mente *em virtude de algum outro motivo*. Searle diz que algumas máquinas têm mentes: os seres humanos são máquinas biológicas com mentes. De acordo com Searle, os cérebros humanos podem ou não estar executando algo como um programa de IA, mas se estiverem essa não é a razão pela qual eles são mentes. É preciso mais para construir uma mente — de acordo com Searle, algo equivalente ao poder causal de neurônios individuais. Ficou indeterminada a questão desses poderes. Deve-se notar, no entanto, que os neurônios evoluíram para cumprir papéis *funcionais* — criaturas com neurônios estão aprendendo e decidindo muito antes de a consciência haver aparecido em cena. Seria

uma coincidência notável se, por acaso, tais neurônios gerassem a consciência por causa de alguns poderes causais que são irrelevantes para as suas capacidades funcionais; afinal de contas, são as capacidades funcionais que determinam a sobrevivência do organismo.

No caso do quarto chinês, Searle confia na intuição e não na prova: basta olhar para o quarto; o que há para ser uma mente? Mas se poderia argumentar da mesma forma sobre o cérebro: olhe para esse conjunto de células (ou átomos) operando cegamente de acordo com as leis da bioquímica (ou da física) — o que existe aí para ser uma mente? Por que um pedaço do cérebro pode ser uma mente enquanto um pedaço de fígado não pode? Isso permanece o grande mistério.

26.2.4 Consciência, qualia e a lacuna explicativa

Uma questão central em todos os debates sobre a IA forte — o elefante na sala de debates, assim dizendo — é a questão da **consciência**. A consciência, muitas vezes, é quebrada em aspectos como a compreensão e o autoconhecimento. O aspecto que vamos focar é o da *experiência subjetiva*: porque é que ela se sente como algo que tem determinados estados cerebrais (por exemplo, ao comer um hambúrguer), considerando que ela presumivelmente não sente como outra coisa que tenha outro estado físico (por exemplo, como sendo uma rocha). O termo técnico para a natureza intrínseca das experiências é ***qualia*** (da palavra latina que significa "tais coisas").

As *qualia* apresentam um desafio para a razão funcionalista da mente, pois *qualia* diferentes poderiam estar envolvidas no que de outra maneira seriam processos causais isomórficos. Considere, por exemplo, a experiência de pensamento do **espectro invertido**, que a experiência subjetiva da pessoa *X* ao ver objetos vermelhos é a mesma experiência que o resto de nós experimenta quando vemos objetos verdes e vice-versa. *X* ainda chama objetos vermelhos de "vermelho", para em semáforos vermelhos e concorda que a vermelhidão dos semáforos vermelhos é mais intensa do que a vermelhidão do pôr do sol. No entanto, a experiência subjetiva de *X* é apenas diferente.

As *qualia* são desafiadoras, não apenas para o funcionalismo, mas para toda a ciência. Suponha, por causa do argumento, que tenhamos concluído o processo de pesquisa científica sobre o cérebro — descobrimos que o processo neural P_{12} no neurônio N_{177} transforma a molécula *A* em *B*, e assim por diante. Simplesmente não há forma de raciocínio aceita atualmente que conduza a partir de tais achados à conclusão de que a entidade que possui esses neurônios tenha qualquer experiência subjetiva particular. Esse **hiato explicativo** levou alguns filósofos a concluírem que os seres humanos são simplesmente incapazes de formar uma compreensão adequada da sua própria consciência. Outros, notavelmente Daniel Dennett (1991), evitaram o hiato, negando a existência das *qualia*, atribuindo a elas uma confusão filosófica.

Turing admite que a questão da consciência seja uma tarefa difícil, mas nega que tenha muita relevância para a prática de IA: "Não quero dar a impressão de que acho que não há mistério sobre a consciência (…) Mas eu não acho que esses mistérios precisem ser resolvidos necessariamente antes que possamos responder à pergunta com a qual estamos preocupados neste artigo." Concordamos com Turing — estamos interessados na criação de programas que se comportem de forma inteligente. O projeto adicional de torná-los conscientes não é o que estamos preparados para assumir nem

aquele cujo sucesso seríamos capazes de determinar.

26.3 A ÉTICA E OS RISCOS DE DESENVOLVER A INTELIGÊNCIA ARTIFICIAL

Até agora, concentramos nossa atenção no fato de *podermos* ou não desenvolver a IA, mas também devemos considerar se *devemos* ou não fazer isso. Se os efeitos da tecnologia de IA tiverem maior probabilidade de serem negativos do que positivos, será uma questão de responsabilidade moral dos trabalhadores no campo redirecionar sua pesquisa. Muitas tecnologias novas tiveram efeitos colaterais negativos não pretendidos: a fissão nuclear trouxe Chernobyl e a ameaça de destruição global; o motor de combustão interna trouxe a poluição do ar, o aquecimento global e a pavimentação do paraíso. Em certo sentido, os automóveis são robôs que conquistaram o mundo tornando-se indispensáveis.

Todos os cientistas e engenheiros enfrentam considerações éticas sobre como devem agir no trabalho, que projetos devem ou não ser realizados e de que maneira devem ser tratados. Existe até mesmo um manual sobre a *Ethics of Computing* (Berleur e Brunnstein, 2001). Entretanto, a IA parece trazer alguns problemas novos, por exemplo, com a necessidade de construir pontes que não caiam:

- As pessoas poderiam perder seus empregos para a automação.
- As pessoas poderiam ter muito (ou pouco) tempo de lazer.
- As pessoas poderiam perder seu sentido de identidade.
- Sistemas de IA poderiam ser utilizados para fins indesejáveis.
- O uso de sistemas de IA poderia resultar na perda de responsabilidade.
- O sucesso da IA poderia significar o fim da raça humana.

Examinaremos cada uma dessas questões separadamente.

As pessoas poderiam perder seus empregos para a automação. A economia industrial moderna se tornou dependente dos computadores em geral e de alguns programas de IA em particular. Por exemplo, grande parte da economia, especialmente nos Estados Unidos, depende da disponibilidade de crédito ao consumidor. Aplicações de cartão de crédito, aprovações de débitos e detecção de fraudes são feitas agora por programas de IA. Alguém poderia argumentar que milhares de trabalhadores foram demitidos por esses programas de IA, mas, de fato, se não houvesse os programas de IA esses trabalhos não existiriam porque o trabalho humano adicionaria um custo inaceitável às transações. Até agora, a automação por meio da tecnologia de IA criou mais empregos do que eliminou, e criou empregos mais interessantes e com remuneração mais elevada. Agora que o programa de IA canônico é um “agente inteligente” projetado para auxiliar um ser humano, a perda de empregos é uma preocupação menor do que era quando a IA se concentrava em “sistemas especialistas”, projetados para substituir os seres humanos. Mas alguns pesquisadores acreditam que fazer o trabalho completo é o objetivo certo para a IA. Na reflexão sobre o 25º aniversário da AAAI, Nils Nilsson (2005) definiu como desafio a criação de IA em nível humano, que poderia passar em um teste de emprego, em vez do teste de Turing — um robô que pode aprender a fazer qualquer um

de uma série de trabalhos. Podemos acabar em um futuro no qual o desemprego seja alto, mas até mesmo os desempregados servirão como gerentes de seus próprios quadros de trabalhadores robôs.

As pessoas poderiam ter muito (ou pouco) tempo de lazer. Alvin Toffler escreveu em *O choque do futuro* (1970): "A semana de trabalho foi reduzida em 50% desde a virada do século. Não é difícil prever que ela será novamente reduzida à metade por volta de 2000." Arthur C. Clarke (1968b) escreveu que as pessoas em 2001 poderiam estar "diante de um futuro de absoluto enfado, em que o principal problema da vida será decidir que canal selecionar dentre várias centenas de canais de TV". A única dessas previsões que chegou perto de se realizar é o número de canais de TV (Springsteen, 1992). Em vez disso, as pessoas que trabalham em indústrias que fazem uso intensivo do conhecimento descobriram que elas próprias faziam parte de um sistema computadorizado integrado que opera 24 horas por dia; para se manterem atualizadas, elas foram forçadas a trabalhar por turnos *mais longos*. Em uma economia industrial, as recompensas são aproximadamente proporcionais ao tempo investido; trabalhar 10% a mais tenderia a significar um aumento de 10% na renda. Em uma economia de informações marcada por comunicação em alta largura de banda e fácil replicação da propriedade intelectual (o que Frank e Cook (1996) chamam de "sociedade em que o vencedor leva tudo"), existe uma grande recompensa por ser ligeiramente melhor que a concorrência; trabalhar 10% a mais poderia significar um aumento de 100% na renda. Assim, existe pressão crescente sobre todo mundo para trabalhar mais. A IA aumenta o ritmo da inovação tecnológica e, desse modo, contribui para essa tendência global, mas a IA também mantém a promessa de nos oferecer mais tempo livre e de deixar nossos agentes automatizados cuidarem de tudo por algum tempo. Tim Ferriss (2007) recomenda o uso de automação e terceirização para atingir quatro horas de trabalho por semana.

As pessoas poderiam perder seu sentido de identidade. Em *Computer Power and Human Reason*, Weizenbaum (1976), o autor do programa ELIZA, destaca algumas das ameaças potenciais que a IA representa para a sociedade. Um dos principais argumentos de Weizenbaum é que a pesquisa de IA torna possível a ideia de que os seres humanos são autômatos — uma ideia que resulta em perda de autonomia ou até de humanidade. Observamos que a ideia tem estado presente há muito mais tempo que a IA, desde a época de *L'Homme Machine* (La Mettrie, 1748). Também observamos que a humanidade sobreviveu a outros retrocessos em nosso senso de identidade: *De Revolutionibus Orbium Coelestium* (Copérnico, 1543) afastou a Terra do centro do sistema solar, enquanto *A descendência do homem* (Darwin, 1871) colocou o *Homo sapiens* no mesmo nível das outras espécies. A IA, se for amplamente bem-sucedida, poderá ser pelo menos tão ameaçadora para as suposições morais da sociedade do século XXI como a teoria da evolução de Darwin foi para as suposições morais do século XIX.

Sistemas de inteligência artificial poderiam ser utilizados para fins indesejáveis. Tecnologias avançadas têm sido utilizadas muitas vezes pelos poderosos para reprimir os seus rivais. Como o matemático G. H. Hardy escreveu (Hardy, 1940): "Diz-se que a ciência é útil se o seu desenvolvimento tende a acentuar as desigualdades existentes na distribuição da riqueza ou mais diretamente promove a destruição da vida humana." Isso vale para todas as ciências, a IA não é uma exceção. Sistemas autônomos de IA agora são comuns no campo de batalha; o exército dos Estados Unidos implantou mais de 5.000 aeronaves autônomas e 12.000 veículos autônomos de chão no Iraque (Singer, 2009). Uma teoria moral afirma que os robôs militares são como uma armadura

medieval levada ao seu extremo lógico: ninguém tem objeções morais a um soldado querer usar um capacete quando está sendo atacado por inimigos numerosos, irritados, brandindo o machado, e um robô teleoperado é como uma forma muito segura de armadura. Por outro lado, armas robóticas representam riscos adicionais. À medida que a decisão humana é retirada do circuito de disparo, os robôs podem acabar tomando decisões que levam à matança de civis inocentes. Em escala maior, a posse de robôs poderosos (como a posse de capacetes resistentes) pode dar um excesso de confiança à nação, fazendo-a entrar em guerra de forma mais imprudente do que o necessário. Na maioria das guerras, pelo menos uma parte está confiante em suas habilidades militares — caso contrário, o conflito teria sido resolvido pacificamente.

Weizenbaum (1976) também assinalou que a tecnologia de reconhecimento da fala poderia levar à disseminação da espionagem e, portanto, a uma perda das liberdades civis. Ele não previu um mundo com ameaças terroristas que mudariam a proporção de vigilância que as pessoas estariam dispostas a aceitar, mas reconheceu corretamente que a IA tem potencial para produzir vigilância em massa. Sua previsão talvez se torne verdadeira: o Reino Unido agora tem uma extensa rede de vigilância por câmeras, e outros países monitoram rotineiramente o tráfego da Web e as chamadas telefônicas. Alguns admitem que a computadorização leva a uma perda de privacidade — o CEO da Sun Microsystems, Scott McNealy, disse: "De qualquer maneira, você tem privacidade zero. Conforme-se."

David Brin (1998) argumenta que a perda de privacidade é inevitável, e a forma de combater a assimetria de poder do Estado sobre o indivíduo é tornar a vigilância acessível a todos os cidadãos. Etzioni (2004) defende um balanceamento entre a privacidade e a segurança, direitos individuais e comunidade.

O uso de sistemas de IA poderia resultar na perda de responsabilidade. Na atmosfera litigiosa que prevalece nos Estados Unidos, a responsabilidade legal se torna uma questão importante. Quando um médico conta com o julgamento de um sistema especialista médico para um diagnóstico, quem será o responsável caso o diagnóstico esteja errado? Felizmente, em parte devido à crescente influência dos métodos de teoria da decisão na medicina, agora se admite que a negligência não pode ser mostrada se o médico executar procedimentos médicos que tenham alta utilidade *esperada*, mesmo que o resultado *real* seja catastrófico para o paciente. A pergunta então deve ser: "Quem será o responsável se o diagnóstico for irracional?" Até agora, os tribunais têm considerado que os sistemas especialistas médicos desempenham o mesmo papel dos livros didáticos e dos livros de referência de medicina; os médicos são responsáveis pela compreensão do raciocínio por trás de qualquer decisão e pelo uso de seu próprio critério para decidir se devem aceitar as recomendações do sistema. Portanto, no projeto de sistemas especialistas médicos como agentes, as ações devem ser vistas não como algo que afeta diretamente o paciente, mas como influências no comportamento do médico. Se os sistemas especialistas se tornarem mais confiáveis e precisos que os diagnósticos humanos, os médicos poderão se tornar legalmente responsáveis se *não* usarem as recomendações de um sistema especialista. Atul Gawande (2002) explora essa premissa.

Questões semelhantes estão começando a surgir em relação ao uso de agentes inteligentes na Internet. Foi feito algum progresso na incorporação de restrições a agentes inteligentes de forma que eles não possam, por exemplo, danificar os arquivos de outros usuários (Weld e Etzioni, 1994). O problema é ampliado quando circula dinheiro. Se forem efetuadas transações monetárias "em nome

de alguém" por um agente inteligente, essa pessoa será responsável pelos débitos que surgirem? Seria possível um agente inteligente ter recursos próprios e executar transações eletrônicas em seu benefício? Até agora, essas perguntas não parecem estar bem compreendidas. Até onde sabemos, nenhum programa recebeu *status* legal como indivíduo para fins de transações financeiras; no momento, parece irracional fazê-lo. Os programas também não são considerados "motoristas" para fins de imposição das leis de trânsito em rodovias reais. Pelo menos segundo as leis da Califórnia, parece não haver nenhuma sanção legal para impedir um veículo automatizado de exceder os limites de velocidade, embora o projetista do mecanismo de controle do veículo seja responsabilizado no caso de um acidente. Como ocorre com a tecnologia da reprodução humana, a lei ainda não consegue acompanhar os novos desenvolvimentos.

O sucesso da IA poderia significar o fim da raça humana. Quase toda tecnologia tem potencial para causar danos em mãos erradas; porém, no caso da IA e da robótica, temos um novo problema, pois as mãos erradas podem pertencer à própria tecnologia. Incontáveis histórias de ficção científica advertiram sobre robôs ou *cyborgs* (robôs humanos) que causam devastação. Exemplos antigos incluem o livro de Mary Shelley, *Frankenstein* (1818),[5] e a peça de Karel Capek, *R.U.R* (1921), em que os robôs conquistam o mundo. No cinema, temos *O exterminador do futuro* (1984), que combina os clichês de robôs que conquistam o mundo com a viagem no tempo, e *Matrix* (1999), que combina robôs que conquistaram o mundo com cérebro em uma cuba.

Na maioria dos casos, parece que os robôs são os protagonistas de tantas histórias de conquista do mundo porque representam o desconhecido, da mesma forma que as bruxas e os fantasmas desde os contos antigos, ou os marcianos de *A guerra dos mundos* (Wells, 1898). A questão é se um sistema de IA representa um risco maior do que o software tradicional. Vamos olhar para três fontes de risco.

Primeiro, a estimativa do estado do sistema de IA pode estar incorreta, fazendo com que as coisas deem errado. Por exemplo, um carro autônomo pode estimar incorretamente a posição de um carro na faixa adjacente, levando a um acidente que poderia matar os ocupantes. Mais sério, um sistema de defesa de míssil pode detectar erroneamente um ataque e lançar um contra-ataque, levando bilhões à morte. Esses riscos não são realmente riscos dos sistemas de IA — em ambos os casos, o mesmo erro poderia facilmente ser feito por um ser humano ou por um computador. A forma correta de mitigar esses riscos é a concepção de um sistema com uma série de pesos e contrapesos para que um único erro de estimativa de estado não se propague através do sistema não verificado.

Segundo, especificar a função utilidade correta para um sistema de IA maximizar não é tão fácil. Por exemplo, poderíamos propor uma função utilidade projetada para *minimizar o sofrimento humano,* expressa como uma função de recompensa aditiva ao longo do tempo, como no Capítulo 17. Dada a forma como os seres humanos são, no entanto, vamos sempre encontrar uma maneira de sofrer, mesmo no paraíso; por isso, a decisão ótima para o sistema de IA é terminar com a raça humana o mais rápido possível — sem seres humanos, sem sofrimento. Com sistemas de IA, então, precisamos ter muito cuidado com as solicitações, enquanto os humanos não têm nenhum problema em perceber que a função utilidade proposta não pode ser tomada literalmente. Por outro lado, os computadores não precisam ser corrompidos pelos comportamentos irracionais descritos no Capítulo 16. Os seres humanos, às vezes, usam sua inteligência de maneira agressiva porque, por natureza, os seres humanos têm algumas tendências agressivas, devido à seleção natural. As máquinas que construímos não precisam ser agressivas inatamente, a menos que decidamos construí-las dessa

forma (ou a menos que surjam como o produto final de um projeto de mecanismo que estimule o comportamento agressivo). Felizmente, existem técnicas, como a aprendizagem de aprendizado, que nos permitem especificar uma função utilidade como exemplo. Pode-se esperar que um robô que seja inteligente o suficiente para descobrir como terminar com a raça humana também seja inteligente o suficiente para descobrir que não era a função utilidade pretendida.

Em terceiro lugar, a função de aprendizagem do sistema de IA pode fazer com que evolua para um sistema com comportamento indesejado. Esse cenário é o mais sério e exclusivo para os sistemas de IA; então, vamos abrangê-lo com mais profundidade. I. J. Good escreveu (1965):

> Vamos definir uma **máquina ultrainteligente** como uma máquina que pode superar de longe todas as atividades intelectuais de qualquer homem inteligente. Tendo em vista que o projeto de máquinas é uma dessas atividades intelectuais, uma máquina ultrainteligente poderia projetar máquinas ainda melhores; sem dúvida, haveria uma "explosão de inteligência" e a inteligência do homem ficaria para trás. Desse modo, a primeira máquina ultrainteligente é a *última* invenção que o homem precisaria realizar, desde que a máquina seja dócil o suficiente para nos dizer como mantê-la sob controle.

A "explosão de inteligência" também foi chamada **singularidade tecnológica** pelo professor de matemática e autor de ficção científica Vernor Vinge, que escreveu (1993): "Dentro de trinta anos, teremos os meios tecnológicos para criar uma inteligência super-humana. Logo depois, a era humana estará terminada." Good e Vinge (e muitos outros) observam corretamente que a curva do progresso tecnológico está crescendo exponencialmente no momento (considere a lei de Moore). Porém, é fácil extrapolar e mostrar que a curva continuará na direção de uma singularidade com crescimento próximo ao infinito. Até agora, todas as outras tecnologias seguiram uma curva em forma de S, em que o crescimento exponencial eventualmente diminui. Às vezes, as novas tecnologias entram quando as antigas caem em desuso, às vezes atingimos limites rígidos. Com menos de um século de história de alta tecnologia para seguir em frente, é difícil extrapolar centenas de anos à frente.

Observe que o conceito de máquinas ultrainteligentes assume que a inteligência é um atributo especialmente importante e, se você tiver inteligência suficiente, todos os problemas podem ser resolvidos. Mas sabemos que há limites sobre computabilidade e complexidade computacional. Se o problema da definição de máquinas ultrainteligentes (ou mesmo a aproximações delas) cair na classe de, digamos, problemas NEXPTIME-completos e se não houver atalhos heurísticos, mesmo o progresso exponencial da tecnologia não vai ajudar — a velocidade da luz coloca um limite superior estrito no quanto de computação pode ser feito; problemas além desse limite não serão resolvidos. Ainda não sabemos onde estão esses limites superiores.

Vinge está preocupado e assustado com a próxima singularidade, mas outros cientistas da computação e futuristas a apreciam. Hans Moravec (2000) nos encoraja a dar todas as vantagens para nossas "mentes infantis", os robôs que criamos, que podem nos ultrapassar em inteligência. Há até mesmo uma nova palavra — **transumanismo** — para o movimento social ativo que antecipa esse futuro em que os seres humanos estão imersos ou substituídos por invenções robóticas e biotecnológicas. Basta dizer que essas questões representam um desafio para a maioria dos teóricos da moral, que tomam a preservação da vida e da espécie humana como algo bom. Ray Kurzweil é atualmente o defensor mais visível sob o ponto de vista da singularidade, escrevendo em *The*

Singularity Is Near (2005):

> A singularidade nos permitirá transcender essas limitações de nosso corpo biológico e do cérebro. Vamos ganhar poder sobre nossos destinos. Nossa mortalidade estará em nossas próprias mãos. Seremos capazes de viver por tanto tempo quanto quisermos (uma declaração sutilmente diferente de dizer que vamos viver para sempre). Vamos compreender o pensamento humano totalmente e vamos estender e expandir muito o seu alcance. Até o final deste século, a porção não biológica de nossa inteligência, sem ajuda, será trilhões de trilhões de vezes mais poderosa que a inteligência humana.

Kurzweil também observa os perigos potenciais, escrevendo: "Mas a singularidade também vai ampliar a capacidade de agir sobre nossas inclinações destrutivas, então a sua história completa ainda não foi escrita."

Se as máquinas ultrainteligentes são uma possibilidade, nós, seres humanos, faríamos bem em ter certeza que projetamos os seus antecessores, de tal forma que eles projetem a sim mesmos para nos tratar bem. O escritor de ficção científica Isaac Asimov (1942) foi o primeiro a abordar essa questão, com suas três leis da robótica:

1. Um robô não pode ferir um ser humano ou, por omissão, permitir que um ser humano sofra algum mal.
2. Um robô deve obedecer às ordens que lhe sejam dadas por seres humanos, exceto nos casos em que tais ordens contrariem a primeira lei.
3. Um robô deve proteger sua própria existência desde que tal proteção não entre em conflito com a primeira ou a segunda lei.

Essas leis parecem razoáveis, pelo menos para nós humanos.[6] Mas o desafio é como implementar essas leis. Na história de Asimov, chamada *Roundabout* um robô é enviado para buscar selênio. Mais tarde, encontra-se o robô vagando em círculo em torno da fonte de selênio. Cada vez que se dirige para a fonte, ele sente um perigo, e a terceira lei faz com que ele desvie. Mas, a cada vez que desvia, o perigo se afasta, e o poder da segunda lei assume, fazendo com que ele vá de volta ao selênio. O conjunto de pontos que definem o ponto de equilíbrio entre as duas leis define um círculo. Isso sugere que as leis não são lógicas absolutas, mas ponderadas umas contra as outras, com um peso maior para as leis anteriores. Asimov estava pensando provavelmente em uma arquitetura baseada em teoria de controle — talvez uma combinação linear de fatores, enquanto hoje a arquitetura mais provável seria um agente de raciocínio probabilístico que raciocine sobre distribuições de probabilidade dos resultados e maximize a utilidade, conforme definido pelas três leis. Mas, presumivelmente, não queremos que nossos robôs evitem que um ser humano atravesse a rua por causa da possibilidade de dano diferente de zero. Isso significa que a utilidade negativa para danos a um ser humano deve ser muito maior do que desobedecer, mas que cada uma das utilidades é finita, não infinita.

Yudkowsky (2008) entra em mais detalhes sobre como projetar um **IA amigável**. Afirma que a afabilidade (um desejo de não prejudicar os seres humanos) deve ser concebida desde o início, mas que os projetistas devem reconhecer que tanto os seus próprios projetos podem ser falhos como que

o robô vai aprender e evoluir com o tempo. Assim, o desafio é do projeto do mecanismo — definir um mecanismo para a evolução de sistemas de inteligência artificial com um sistema de pesos e contrapesos, e fornecer funções utilidade ao sistema que permanecerão amigáveis em face de tais mudanças.

Não podemos fornecer a um programa apenas uma função utilidade estática porque as circunstâncias e as respostas desejadas às circunstâncias mudam com o tempo. Por exemplo, se a tecnologia tivesse permitido que criássemos um agente de IA superpoderoso em 1800 e o dotássemos com a moral vigente da época, ele estaria lutando hoje para restabelecer a escravidão e abolir o direito das mulheres ao voto. Por outro lado, se construirmos um agente de IA hoje e dissermos a ele para evoluir a sua função utilidade, como poderemos assegurar que ele não raciocine que "os seres humanos pensam que é moral matar insetos irritantes, em parte parte porque o cérebro dos insetos são tão primitivos. Mas o cérebro humano é primitivo em comparação com os meus poderes, por isso deve ser moral matar os seres humanos".

Omohundro (2008) levantou a hipótese de que mesmo um programa de xadrez inócuo poderia representar um risco para a sociedade. Da mesma forma, Marvin Minsky, sugeriu uma vez que um programa de IA projetado para resolver a hipótese de Riemann poderia acabar tomando posse de todos os recursos da Terra para construir supercomputadores mais poderosos para ajudar a alcançar seu objetivo. A moral é que, mesmo se você só deseja que o seu programa jogue xadrez ou demonstre teoremas, se lhe der a capacidade de aprender e alterar a si mesmo são necessárias salvaguardas. Omohundro conclui que "as estruturas sociais que fazem com que os indivíduos arquem com o custo de suas externalidades negativas tem um longo caminho para garantir um futuro estável e positivo". Essa parece ser uma ideia excelente para a sociedade em geral, independentemente da possibilidade de máquinas ultrainteligentes.

Devemos notar que a ideia de salvaguardas contra a mudança na função utilidade não é nova. Na *Odisseia*, Homero (c. 700 a.C.) descreveu o encontro de Ulisses com as sereias, cuja música era tão sedutora que obrigava os marinheiros a se lançarem ao mar. Sabendo que isso teria o mesmo efeito sobre ele, Ulisses ordenou à sua tripulação que o amarrasse ao mastro para que ele não pudesse realizar o ato autodestrutivo. É interessante pensar como salvaguardas semelhantes poderiam ser construídas em sistemas de IA.

Por fim, vamos considerar o ponto de vista do robô. Se os robôs se tornarem conscientes, poderia ser imoral tratá-los como meras "máquinas" (por exemplo, para segregá-los). Os próprios robôs também devem agir de acordo com a moralidade — precisaríamos programá-los com uma teoria do que é certo e errado. Os escritores de ficção científica atacaram a questão dos direitos dos robôs. O conhecido filme *A.I.* (Spielberg, 2001) se baseou em uma história de Brian Aldiss sobre um robô inteligente que foi programado para acreditar que era humano e não consegue entender seu abandono eventual pela mãe e proprietária. A história (e o filme) argumentam a favor da necessidade de direitos civis para os robôs.

26.4 RESUMO

Este capítulo tratou as seguintes questões:

- Os filósofos utilizam a expressão **IA fraca** para representar a hipótese de que as máquinas talvez possam se comportar com inteligência, e a expressão **IA forte** para representar a hipótese de que tais máquinas contariam com mentes reais (em oposição a mentes simuladas).
- Alan Turing rejeitou a pergunta "As máquinas podem pensar?" e a substituiu por um teste comportamental. Ele antecipou muitas objeções à possibilidade de máquinas pensantes. Poucos pesquisadores de IA prestam atenção ao teste de Turing, preferindo se concentrar no desempenho de seus sistemas em tarefas práticas, e não na habilidade para imitar os seres humanos.
- Há concordância geral em tempos modernos sobre o fato de que os estados mentais são estados cerebrais.
- Os argumentos a favor e contra a IA forte são inconclusivos. Poucos pesquisadores importantes de IA acreditam que venha a surgir alguma conclusão significativa no resultado do debate.
- A consciência continua a ser um mistério.
- Identificamos oito ameaças potenciais à sociedade representadas pela IA e pela tecnologia inter-relacionada. Concluímos que algumas ameaças sejam improváveis ou pouco diferem das ameaças postadas por tecnologias "não inteligentes". Uma ameaça em particular é digna de maiores considerações: que as máquinas ultrainteligentes podem levar a um futuro que é muito diferente de hoje — podemos não gostar dele e, nesse ponto, podemos não ter mais escolha. Tais considerações conduzem inevitavelmente à conclusão de que é preciso pesar cuidadosamente, e logo, as consequências possíveis da pesquisa em IA.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

O capítulo apresentou fontes de várias respostas à tese de Turing de 1950 e dos principais críticos da IA fraca. Embora tenha se tornado moda na era pós-rede neural ridicularizar abordagens simbólicas, nem todos os filósofos são críticos da GOFAI. Alguns são, de fato, defensores ardorosos e até mesmo praticantes. Zenon Pylyshyn (1984) argumentou que a cognição pode ser mais bem compreendida através de um modelo computacional, não só em princípio, mas também como forma de realização de pesquisa no momento, e refutou especificamente as críticas de Dreyfus do modelo computacional da cognição humana (Pylyshyn, 1974). Gilbert Harman (1983), ao analisar a revisão de crença, faz conexões com a pesquisa em IA sobre sistemas de manutenção de verdade. Michael Bratman aplicou seu modelo de psicologia humana "crença-desejo-intenção" (Bratman, 1987) à pesquisa em IA sobre planejamento (Bratman, 1992). No fim extremo da IA forte, Aaron Sloman (1978, p. xiii) chegou mesmo a descrever como "racista" a reivindicação de Joseph Weizenbaum (1976) de que as máquinas inteligentes não podem ser consideradas como pessoas.

Os defensores da importância do corpo na cognição incluem os filósofos Merleau-Ponty, cuja *Phenomenology of Perception* (1945) salientou a importância do corpo e da interpretação subjetiva da realidade oferecida pelos nossos sentidos, e Heidegger, que em *Being and Time* (1927) perguntou o que significa realmente ser um agente, e criticou toda a história da filosofia por presumir essa noção. Na era do computador, Alva Noe (2009) e Andy Clark (1998, 2008) propõem que o nosso cérebro forme uma representação bastante minimalista do mundo, e usam o próprio mundo baseado no momento exato para manter a ilusão de um modelo interno detalhado, o uso de adereços no mundo

(como papel e lápis, bem como computadores) para aumentar a capacidade da mente. Pfeifer *et al.* (2006) e Lakoff e Johnson (1999) apresentaram argumentos sobre como o corpo ajuda a cognição da forma.

A natureza da mente foi um tópico-padrão da teorização filosófica desde a Antiguidade até o presente. Em sua obra *Phaedo*, Platão considerou e rejeitou especificamente a ideia de que a mente poderia ser uma "harmonização" ou um padrão de organização das partes do corpo, um ponto de vista que se aproxima da visão funcionalista da moderna filosofia da mente. Em vez disso, ele afirmou que a mente tinha de ser uma espécie de alma imortal e imaterial, separada do corpo e diferente deste em substância — o ponto de vista do dualismo. Aristóteles distinguiu uma variedade de almas (em grego, ψνχη) em seres vivos, alguns dos quais, pelo menos, ele descreveu de modo funcionalista (veja em Nussbaum (1978) mais informações sobre o funcionalismo de Aristóteles).

Descartes é notório por sua visão dualista da mente humana mas, ironicamente, sua influência histórica tendia ao mecanicismo e ao fisicalismo. Ele concebeu explicitamente animais como autômatos e antecipou o teste de Turing, escrevendo que "não é concebível que [uma máquina] deva produzir combinações diferentes de palavras para dar uma resposta apropriada ao que for dito em sua presença, como até mesmo o mais estúpido dos homens pode fazer" (Descartes, 1637). A defesa calorosa de Descartes, do ponto de vista de animais como autômatos, teve na realidade o efeito de facilitar também a concepção de humanos como autômatos, embora ele próprio não chegasse a esse ponto. O livro *L'Homme Machine* (La Mettrie, 1748) argumentava de forma explícita que os seres humanos são autômatos.

A filosofia analítica moderna tem aceitado geralmente o fisicalismo, mas a variedade de pontos de vista sobre o conteúdo dos estados mentais é desconcertante. A identificação de estados mentais com os estados do cérebro é geralmente atribuída a Place (1956) e Smart (1959). O debate entre os pontos de vista de conteúdo restrito e amplo dos estados mentais foi desencadeado por Hilary Putnam (1975), que introduziu as chamadas **terras gêmeas** (em vez do cérebro em uma cuba, como no capítulo) como um dispositivo para gerar estados cerebrais com conteúdo idêntico (amplo) diferente.

O funcionalismo é a filosofia da mente sugerida mais naturalmente pela IA. A ideia de que os estados mentais correspondem às classes de estados cerebrais definidos funcionalmente é devida a Putnam (1960, 1967) e Lewis (1966, 1980). Talvez o defensor mais forte do funcionalismo seja Daniel Dennett, cujo trabalho intitulado ambiciosamente *Consciousness Explained* (Dennett, 1991) tem atraído muitas tentativas de refutações. Metzinger (2009) argumenta que isso não existe como objetivo em *si,* que a consciência é o aspecto subjetivo de um mundo. O argumento do espectro invertido sobre *qualia* foi introduzido por John Locke (1690). Frank Jackson (1982) desenvolveu um experimento de pensamento influente envolvendo Mary, uma cientista de cor que foi criada em um mundo totalmente em preto e branco. *Quem Vai Ficar com Mary* (Ludlow *et al*., 2004) reuniu vários artigos sobre esse tópico.

Os autores que afirmam que não levam em conta as *qualia* ou "o sentido" dos aspectos de estados mentais (Nagel, 1974) atacaram o funcionalismo. Em vez disso, Searle concentrou-se na incapacidade do funcionalismo alegada de explicar a intencionalidade (Searle, 1980, 1984, 1992). Churchland e Churchland (1982) refutaram esses dois tipos de críticas. O quarto chinês foi debatido interminavelmente (Searle, 1980, 1990; Preston e Bishop, 2002). Vamos mencionar aqui apenas um

trabalho relacionado: a história de ficção científica *They're Made Out of Meat* de Terry Bisson (1990), em que exploradores robóticos alienígenas que visitam a Terra ficam incrédulos ao encontrar seres humanos pensantes cuja mente é feita de carne. Presumivelmente, o alienígena robótico equivalente de Searle acredita que ele pode pensar, devido aos poderes causais especiais de circuitos robóticos; poderes causais que os meros cérebros de carne não possuem.

Questões éticas em IA são anteriores à existência do próprio campo. A ideia da máquina ultrainteligente de I. J. Good (1965) foi prevista uma centena de anos antes por Samuel Butler (1863). Escrito quatro anos após a publicação de *A origem das espécies*, de Darwin, e em um tempo em que as máquinas mais sofisticadas eram motores a vapor, o artigo de Butler *Darwin Among the Machines* imaginou "o desenvolvimento máximo da consciência mecânica" por seleção natural. O tema foi reiterado por George Dyson (1998) em um livro de mesmo título.

A literatura filosófica sobre mente, cérebro e tópicos inter-relacionados é grande, e às vezes difícil de ler sem treinamento adequado sobre a terminologia e os métodos de argumentação empregados. A *Encyclopedia of Philosophy* (Edwards, 1967) é um auxiliar impressionantemente autorizado e muito útil nesse processo. *The Cambridge Dictionary of Philosophy* (Audi, 1999) é um trabalho mais sucinto e mais acessível, e a *Stanford Encyclopedia of Philosophy* oferece muitos artigos excelentes e referências atualizadas. A *MIT Encyclopedia of Cognitive Science* (Wilson e Keil, 1999) abrange a filosofia da mente, bem como a biologia e a psicologia da mente. Existem ainda várias introduções gerais à "questão filosófica da IA" (Boden, 1990; Haugeland, 1985; Copeland, 1993; McCorduck, 2004; Minsky, 2007). *The Behavioral and Brain Sciences*, abreviado como *BBS*, é um periódico importante dedicado a debates filosóficos e científicos sobre IA e neurociência. Tópicos de ética e responsabilidade em IA são focalizados em periódicos como *AI and Society* e *Journal Artificial Intelligence and Law*.

EXERCÍCIOS

26.1 Percorra a lista de Turing de alegadas "inaptidões" das máquinas, identificando quais delas foram alcançadas, quais podem em princípio ser alcançadas por um programa e quais ainda são problemáticas porque exigem estados mentais conscientes.

26.2 Encontre e analise na mídia popular uma consideração sobre um ou mais argumentos de que IA é impossível.

26.3 No argumento de prótese cerebral, é importante poder restaurar o cérebro do indivíduo à condição normal, de tal forma que seu comportamento externo volte a ser o que teria sido se a operação não houvesse ocorrido. O cético poderia objetar de forma razoável que isso exigiria a atualização das propriedades neurofisiológicas dos neurônios relacionadas à experiência consciente, de maneira distinta daquelas propriedades envolvidas no comportamento funcional dos neurônios?

26.4 Suponha que um programa Prolog contendo diversas cláusulas sobre as regras de cidadania britânica seja compilado e execute em um computador normal. Analise os "estados do cérebro" do computador processando conteúdo amplo ou específico.

26.5 Alan Perlis (1982) escreveu: "Um ano dedicado à inteligência artificial é suficiente para

construir uma crença em Deus.” Ele também escreveu em uma carta a Philip Davis que um dos sonhos centrais da ciência da computação é que “através do desempenho dos computadores e seus programas, vamos eliminar qualquer dúvida de que no mundo existe apenas uma distinção química entre os vivos e os não vivos”. Até que ponto o progresso realizado até agora em inteligência artificial lança luz sobre essas questões? Suponha que, em alguma data futura, o esforço em IA tenha sucesso completo, isto é, construiremos agentes inteligentes capazes de realizar qualquer tarefa cognitiva humana em níveis de habilidade humana. Até que ponto isso esclarece essas questões?

26.6 Compare o impacto social da inteligência artificial nos últimos 50 anos com o impacto social da introdução de aparelhos elétricos e o motor de combustão interna nos 50 anos entre 1890 e 1940.

26.7 I. J. Good declarou que a inteligência é a qualidade mais importante e que o desenvolvimento de máquinas ultrainteligentes vai mudar tudo. Um guepardo senciente contra argumentou que “na verdade, a velocidade é mais importante; se pudéssemos construir máquinas muito rápidas, isso mudaria tudo”, e um elefante senciente alegou: “Vocês dois estão errados, o que precisamos é de máquinas muito fortes.” O que você acha desses argumentos?

26.8 Analise as ameaças potenciais da tecnologia da IA à sociedade. Que ameaças são mais sérias e como elas poderiam ser combatidas? Como elas poderiam ser comparadas aos benefícios potenciais?

26.9 Como as ameaças potenciais da tecnologia da IA podem ser comparadas às de outras tecnologias da ciência da computação e à biotecnologia, à nanotecnologia e à tecnologia nuclear?

26.10 Alguns críticos argumentam que a IA é impossível, enquanto outros argumentam que ela é possível *demais* e que máquinas ultrainteligentes representam uma ameaça. Qual dessas objeções é a mais provável em sua opinião? Seria contraditório defender ambas as posições?

---

[1] Por exemplo, a ópera *Coppélia* (1870), o romance *Do Androids Dream of Electric Sheep?* (1968), os filmes *IA* (2001) e *Wall-E* (2008), e a versão da canção de Noel Coward, de 1955, *Let’s Do It: Lest’s Fall in Love* preveem que “provavelmente vamos viver para ver as máquinas fazerem isso”. Ele não conseguiu.

[2] Essa situação pode ser familiar para aqueles que viram o filme *The Matrix* de 1999.

[3] Para comparar, imagine utilizar uma cobaia de “controle” idêntico ao que é fornecido numa operação de placebo.

[4] O fato de as pilhas de papel poderem ser maiores que o planeta inteiro e a geração de respostas poder demorar milhões de anos não tem nenhuma influência sobre a estrutura *lógica* do argumento. Um objetivo do treinamento filosófico é desenvolver um sentido finamente aguçado de quais objeções são pertinentes e quais não são

[5] Quando jovem, Charles Babbage foi influenciado pela leitura de *Frankenstein*.

[6] Um robô pode achar que seja injustiça que um ser humano mate outro em legítima defesa, mas um robô deve sacrificar sua própria vida para salvar um ser humano.

CAPÍTULO 27

IA, presente e futuro

Em que realizamos o inventário de onde estamos e definimos para onde estamos indo, pois isso é sempre algo bom de se fazer antes de prosseguir.

No Capítulo 2, sugerimos que seria útil visualizar a tarefa de IA como a de projetar agentes racionais, isto é, agentes cujas ações maximizam a utilidade esperada considerando suas histórias de percepção. Mostramos que o problema de projeto depende das percepções e das ações disponíveis para o agente, das funções utilidade que o comportamento do agente deve satisfazer e da natureza do ambiente. É possível uma variedade de projetos de agente distintos, variando desde agentes reativos até agentes cognitivos, deliberativos, baseados em conhecimento e em teoria da decisão. Além disso, os componentes desses projetos podem ter várias instanciações diferentes — por exemplo, raciocínio lógico ou probabilístico e representações de estados atômicas, fatoradas ou estruturadas. Os demais capítulos apresentaram os princípios pelos quais esses componentes operam.

Para todos os projetos e componentes de agentes, tem havido enorme progresso, tanto em nossa compreensão científica quanto em nossas competências tecnológicas. Neste capítulo, vamos nos afastar dos detalhes e perguntar: "*Todo esse progresso levará a um agente inteligente de uso geral capaz de se sair bem em ampla variedade de ambientes*?*"* A Seção 27.1 examina os componentes de um agente inteligente para avaliar o que se sabe e o que está faltando. A Seção 27.2 faz o mesmo para a arquitetura de agente global. A Seção 27.3 pergunta em primeiro lugar se projetar agentes racionais é o objetivo correto (a resposta é: "Na realidade, não, mas por ora tudo bem."). Finalmente, a Seção 27.4 examina as consequências do sucesso em nossos esforços.

27.1 COMPONENTES DE AGENTES

O Capítulo 2 apresentou vários projetos de agentes e seus componentes. Para definir o foco de nossa discussão neste capítulo, examinaremos o agente baseado na utilidade, que mostramos novamente na Figura 27.1.

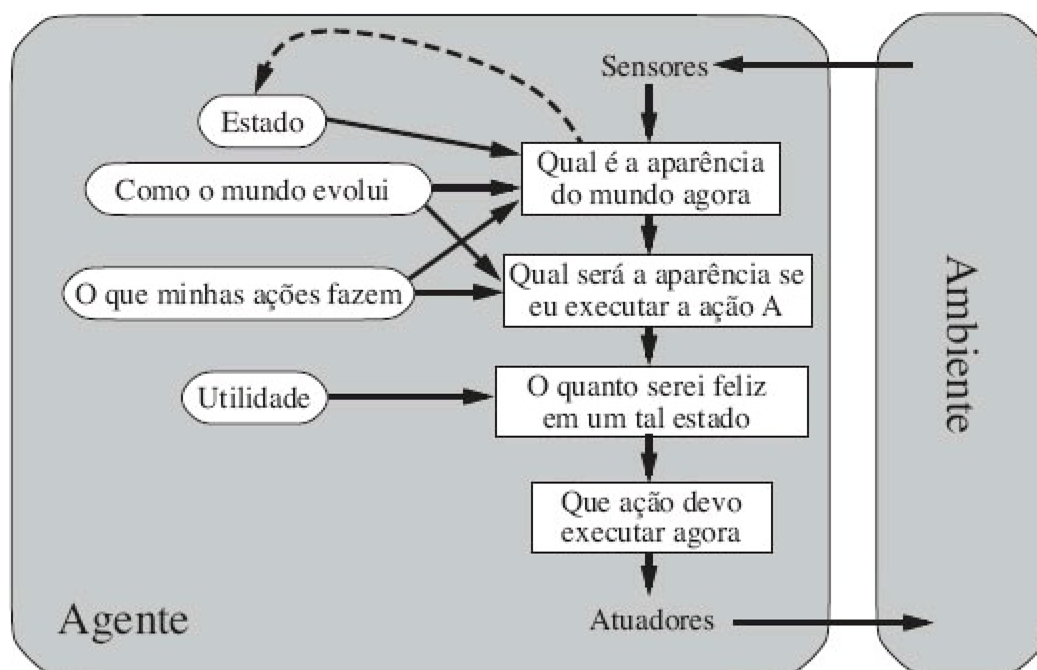


Figura 27.1 Agente de utilidade baseado em modelos, conforme foi apresentado na Figura 2.14.

Quando dotado de um componente de aprendizagem (Figura 2.15), esse é o projeto mais geral do nosso agente. Veremos onde o estado da arte estará situado para cada um dos componentes.

Interação com o ambiente por meio de sensores e atuadores: Para grande parte da história da IA, esse tem sido um visível ponto fraco. Com algumas honrosas exceções, os sistemas de IA foram construídos de tal modo que os seres humanos tinham de fornecer as entradas e interpretar as saídas, enquanto os sistemas robóticos se concentravam em tarefas de baixo nível, nas quais o raciocínio e o planejamento de alto nível estavam quase totalmente ausentes. Em parte, isso foi consequência dos altos custos e do esforço de engenharia exigidos para fazer robôs reais funcionarem. A situação mudou rapidamente nos últimos anos, com a disponibilidade de robôs pré-programáveis. Estes, por sua vez, têm se beneficiado de câmeras de alta resolução CCD pequenas, baratas e compactas, e de motores confiáveis. A tecnologia SMEM (sistemas microeletromecânicos) fornece os acelerômetros miniaturizados, giroscópios e atuadores para um inseto voador artificial (Floreano *et al.*, 2009). Também pode ser possível combinar milhões de dispositivos SMEM para produzir poderosos atuadores macroscópicos.

Assim, vemos que os sistemas de IA estão na iminência de mudar dos sistemas de software primários para sistemas robóticos físicos. O estado da robótica de hoje é mais ou menos comparável ao estado dos computadores pessoais por volta dos anos 1980: naquele tempo, os pesquisadores e amadores podiam fazer experiências com PCs, mas levaria mais uma década antes que eles se tornassem comuns.

Monitorar o estado do mundo: Esse é um dos recursos centrais exigidos de um agente inteligente. Ele requer tanto percepção quanto atualização de representações internas. O Capítulo 4 mostrou como monitorar as representações de estado atômicos; o Capítulo 7 descreveu como fazer isso para as representações de estado decompostas em fatores (proposicionais); o Capítulo 12 estendeu esse controle à lógica de primeira ordem, e o Capítulo 15 descreveu algoritmos de **filtragem** para raciocínio probabilístico em ambientes incertos. Essas ferramentas de filtragem são exigidas quando está envolvida a percepção real (e, portanto, imperfeita). Os algoritmos atuais de filtragem e

percepção podem ser combinados para realizar um trabalho razoável de informar predicados de baixo nível, como "a xícara está sobre a mesa". Detectar ações de alto nível, tais como "O Dr. Russell está tomando uma xícara de chá com o Dr. Norvig enquanto discutem planos para a próxima semana", é mais difícil. Atualmente pode ser feito (veja a Figura 24.25) apenas com a ajuda de exemplos anotados.

Outro problema é que, embora os algoritmos de filtragem aproximada do Capítulo 15 possam lidar com ambientes bastante grandes, ainda estão lidando com uma representação decomposta em fatores — têm variáveis aleatórias, mas não representam explicitamente objetos e relações. A Seção 14.6 explicou como a probabilidade e a lógica de primeira ordem podem ser combinadas para resolver esse problema, e a Seção 14.6.3 mostrou como podemos lidar com a incerteza sobre a identidade dos objetos. Esperamos que a aplicação dessas ideias ao controle de ambientes complexos produza enormes benefícios. No entanto, ainda precisamos encarar a difícil tarefa de definir esquemas de representação geral, reutilizáveis para domínios complexos. Como discutido no Capítulo 12, em geral ainda não sabemos como fazer isso; apenas para domínios isolados e simples. É possível que um novo foco em representações probabilísticas em vez de representações lógicas, juntamente com aprendizagem de máquina agressiva (em vez de codificação manual do conhecimento), permita progressos.

Projetar, avaliar e selecionar futuros cursos de ação: Os requisitos básicos de representação do conhecimento encontrados aqui são os mesmos que existem para controlar o mundo; a principal dificuldade é lidar com cursos de ação — como ter uma conversação ou uma xícara de chá — que eventualmente consistem em milhares ou milhões de etapas primitivas para um agente real. Somente pela imposição de uma **estrutura hierárquica** sobre o comportamento os seres humanos conseguem lidar com tudo. Vimos na Seção 11.2 como utilizar as representações hierárquicas para lidar com problemas dessa escala; além disso, trabalhar em **aprendizagem por reforço hierárquico** tem gerado resultados em combinar algumas dessas ideias com as técnicas para tomada de decisão sob incerteza descritas no Capítulo 17. Os algoritmos para o caso parcialmente observável (POMDPs) estão usando o mesmo estado de representação atômica que utilizamos para os algoritmos de busca do Capítulo 3. Certamente, ainda há muito trabalho a fazer, mas os fundamentos técnicos em grande parte já foram estabelecidos. A Seção 27.2 discute a questão de como controlar a busca por planos efetivos de longo prazo.

Utilidade como expressão de preferências: Em princípio, basear decisões racionais na maximização da utilidade esperada é algo completamente geral e evita muitos dos problemas das abordagens puramente baseadas em metas, como metas conflitantes e realização incerta. No entanto, até agora houve bem pouco trabalho no sentido de construir funções utilidade *realistas* — por exemplo, imagine a teia complexa de interações de preferências que devem ser compreendidas por um agente que opera como auxiliar de escritório para um ser humano.

Tem-se mostrado muito difícil decompor preferências sobre estados complexos do mesmo modo que as redes bayesianas decompõem crenças sobre estados complexos. Uma razão para isso talvez seja o fato de que as preferências sobre estados sejam na realidade *compiladas* a partir de preferências sobre históricos de estados, os quais são descritos por **funções de recompensa** (veja o Capítulo 17). Ainda que a função de recompensa seja simples, a função utilidade correspondente pode ser muito complexa. Isso sugere que a tarefa de engenharia do conhecimento para funções de

recompensa seja adotada com seriedade como um modo de transmitir a nossos agentes aquilo que desejamos que eles façam.

Aprendizagem: Os Capítulos 18 a 21 descreveram como a aprendizagem em um agente pode ser formulada como aprendizagem indutiva (supervisionada, não supervisionada ou baseada em reforço) das funções que constituem os vários componentes do agente. Foram desenvolvidas técnicas lógicas e estatísticas muito poderosas que podem lidar com problemas bastante grandes, com frequência alcançando ou ultrapassando a capacidade humana em muitas tarefas — enquanto estejam lidando com um vocabulário predefinido de características e conceitos. Por outro lado, a aprendizagem de máquina fez bem pouco progresso no importante problema de construir novas representações em níveis de abstração mais altos que o do vocabulário de entrada. Na visão de computador, por exemplo, a aprendizagem de conceitos complexos, tais como *Sala de aula* e *Cafeteria*, ficaria desnecessariamente difícil se o agente fosse forçado a trabalhar a partir de pixels como representação de entrada; em vez disso, o agente deve ser capaz de formar os primeiros conceitos intermediários, como *Escrivaninha e Bandeja*, sem supervisão humana explícita. Aplicam-se considerações semelhantes ao comportamento de aprendizagem: *TomarUmaXícaraDeChá* é uma etapa muito importante de alto nível em muitos planos, mas como ela chega a uma ação de biblioteca que inicialmente contém ações muito mais simples, como *LevantarBraço* e *Engolir*? Talvez algumas das ideias de **redes de crenças profundas** sejam incorporadas — redes bayesianas com múltiplas camadas de variáveis ocultas, como no trabalho de Hinton *et al*. (2006), Hawkins e Blakeslee (2004) e Bengio e LeCun (2007).

Atualmente, a grande maioria das pesquisas de aprendizagem de máquina assume uma representação decomposta em fatores, a função de aprendizagem *h*: $\mathbb{R}^n \rightarrow \mathbb{R}$ para a regressão e *h*: $\mathbb{R}^n \rightarrow \{0, 1\}$ para classificação. Pesquisadores de aprendizagem precisarão adaptar suas técnicas de muito sucesso de representações decompostas em fatores para representações estruturadas, particularmente representações hierárquicas. O trabalho sobre a programação em lógica indutiva do Capítulo 19 é um primeiro passo nessa direção; um próximo passo natural é combinar essas ideias com as linguagens probabilísticas da Seção 14.6.

A não ser que essas sejam entendidas, seremos confrontados pela difícil tarefa de construir à mão grandes bases de conhecimento do senso comum, uma abordagem que não tem se saído bem até agora. Há uma grande promessa no uso da Web como fonte de textos em linguagem natural, imagens e vídeos para servir como base de conhecimento abrangente, mas até agora os algoritmos de aprendizagem de máquina estão limitados pela quantidade de conhecimento organizado que podem extrair dessas fontes.

27.2 ARQUITETURAS DE AGENTES

É natural perguntar: "Qual das arquiteturas de agente do Capítulo 2 um agente deve usar?" A resposta é: "Todas elas!" Já vimos que as respostas reativas são necessárias em situações nas quais o tempo é importante, enquanto a deliberação baseada no conhecimento permite ao agente planejar com antecedência. Um agente completo deve ser capaz de realizar ambas, utilizando uma **arquitetura híbrida**. Uma propriedade importante das arquiteturas híbridas é o fato de que os limites entre

diferentes componentes de decisão não são fixos. Por exemplo, a **compilação** converte continuamente informações declarativas no nível deliberativo em representações mais eficientes, alcançando eventualmente o nível reativo — veja a Figura 27.2 (esse é o propósito da aprendizagem baseada em explanação, discutido no Capítulo 19). Arquiteturas de agentes como SOAR (Laird *et al*., 1987) e THEO (Mitchell, 1990) têm exatamente essa estrutura. Toda vez que resolvem um problema por deliberação explícita, elas preservam uma versão generalizada da solução para uso pelo componente reativo. Um problema menos estudado é a *reversão* desse processo: quando o ambiente se altera, os reflexos aprendidos podem não ser mais apropriados, e o agente deve retornar ao nível deliberativo para produzir novos comportamentos.

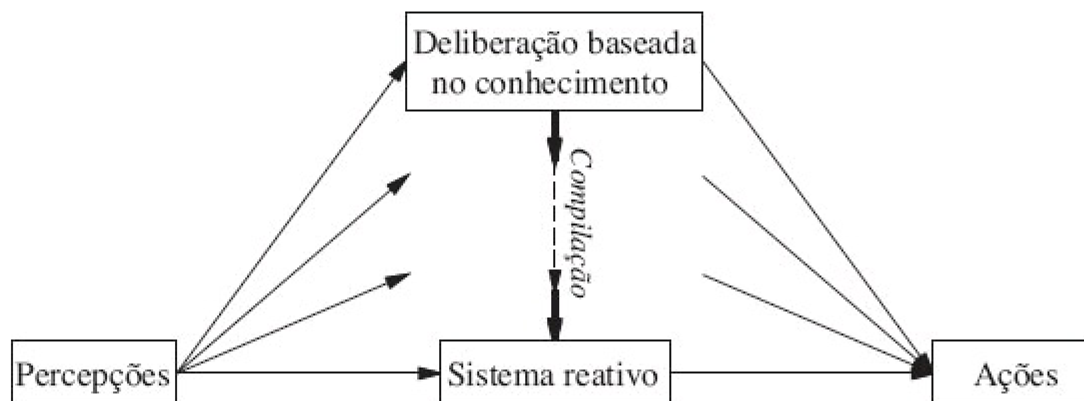


Figura 27.2 A compilação serve para converter a tomada de decisões deliberativa em mecanismos reativos mais eficientes.

Os agentes também precisam de meios para controlar suas próprias deliberações. Eles devem ser capazes de interromper a deliberação quando for exigida uma ação e ser capazes de usar o tempo disponível para deliberação com o objetivo de executar as computações mais vantajosas. Por exemplo, um agente que dirige táxi e vê um acidente à frente tem de decidir em uma fração de segundo se deve frear ou adotar alguma ação evasiva. Ele também deve passar essa fração de segundo pensando nas questões mais importantes, como se as pistas à esquerda e à direita estão vazias e se há um grande caminhão vindo por trás, em vez de se preocupar com o desgaste dos pneus ou onde apanhar o próximo passageiro. Essas questões normalmente são estudadas sob o título de **IA em tempo real**. À medida que os sistemas de IA entram em domínios mais complexos, todos os problemas se tornam problemas de tempo real porque o agente nunca terá tempo suficiente para resolver de forma exata o problema de decisão.

Fica claro que existe necessidade urgente de métodos genéricos de controle de deliberação, em vez de receitas específicas para o que pensar em cada situação. A primeira ideia útil é o emprego de **algoritmos a qualquer tempo** (Dean e Boddy, 1988; Horvitz, 1987). Um algoritmo a qualquer tempo é um algoritmo cuja qualidade de saída melhora gradualmente com o passar do tempo, de forma que ele tenha pronta uma decisão razoável sempre que for interrompido. Tais algoritmos são controlados por um procedimento de decisão de **metanível** que avalia se a computação adicional vale a pena (veja a Seção 3.5.4 para uma breve descrição da tomada de decisão de metanível). Exemplo de um algoritmo a qualquer tempo inclui o aprofundamento iterativo em busca de árvore de jogos e CMMC em redes bayesianas.

A segunda técnica é o **metarraciocínio baseado em teoria da decisão** (Russell e Wefald, 1989,

1991; Horvitz, 1989, Horvitz e Breese, 1996). Esse método aplica a teoria do valor da informação (Capítulo 16) à seleção de computações. O valor de uma computação depende tanto do seu custo (em termos de retardamento da ação) como de seus benefícios (em termos de melhor qualidade da decisão). As técnicas de metarraciocínio podem ser usadas para projetar melhores algoritmos de busca e para garantir que os algoritmos terão a propriedade "a qualquer tempo". É claro que o metarraciocínio é dispendioso, e é possível aplicar métodos de compilação para que os gastos gerais sejam pequenos em comparação com os custos das computações que estão sendo controladas. A aprendizagem por reforço em metanível pode ser outra maneira de adquirir políticas eficazes para controle da deliberação: em essência, os cálculos que levam a melhores decisões são reforçados, enquanto os que acabam por não ter efeito são penalizados. Essa abordagem evita os problemas de miopia no cálculo do valor da informação simples.

O metarraciocínio é apenas um aspecto de uma **arquitetura reflexiva** geral, isto é, uma arquitetura que permite a deliberação sobre as entidades e ações computacionais que ocorrem dentro da própria arquitetura. Um fundamento teórico para arquiteturas reflexivas pode ser construído pela definição de um espaço de estados conjuntos, composto a partir do estado do ambiente e do estado computacional do próprio agente. Podem ser projetados algoritmos de tomada de decisão e de aprendizagem que operem sobre esse espaço de estados conjuntos e, desse modo, sirva para implementar e melhorar as atividades computacionais do agente.

Eventualmente, esperamos que os algoritmos específicos de tarefas como a busca alfa-beta e o encadeamento do agente para trás desapareçam de sistemas de IA, sendo substituídos por métodos gerais que orientam as computações no sentido da geração eficiente de decisões de alta qualidade.

27.3 ESTAMOS INDO NA DIREÇÃO CORRETA?

A seção precedente listou muitos avanços e muitas oportunidades para progresso. No entanto, aonde tudo isso está nos levando? Dreyfus (1992) apresenta a analogia de tentar ir à Lua escalando uma árvore; pode- se informar um progresso constante, durante todo o caminho até a parte superior da árvore. Nesta seção, consideraremos se o caminho atual da IA se parece mais com a escalada de uma árvore ou com a viagem em um foguete.

No Capítulo 1, dissemos que nossa meta era construir agentes que *agissem racionalmente*. Porém, também dissemos que

> ..alcançar a racionalidade perfeita — sempre fazer tudo certo — não é algo viável em ambientes complicados. As demandas computacionais são demasiado elevadas. Porém, na maior parte do livro, adotaremos a hipótese funcional de que a racionalidade perfeita é um bom ponto de partida para a análise.

Agora, chegou o momento de considerarmos mais uma vez qual é exatamente a meta da IA. Queremos construir agentes, mas com que especificação em mente? Aqui estão quatro possibilidades:

Racionalidade perfeita. Um agente perfeitamente racional age a todo instante de forma a

maximizar sua utilidade esperada, dadas as informações que ele adquiriu do ambiente. Vimos que os cálculos necessários para alcançar a racionalidade perfeita na maioria dos ambientes são muito demorados e, assim, a racionalidade perfeita não é uma meta realista.

Racionalidade cautelosa. Essa é a noção de racionalidade que temos usado de forma implícita no projeto de agentes lógicos e de teoria da decisão. Um agente cautelosamente racional retorna *eventualmente* aquela que *teria sido* a escolha racional no início de sua deliberação. Essa é uma propriedade interessante para um sistema exibir, mas, na maioria dos ambientes, a resposta certa no momento errado não tem nenhum valor. Na prática, os projetistas de sistemas de IA são forçados a comprometer a qualidade da decisão para obter desempenho global razoável; infelizmente, a base teórica da racionalidade cautelosa não fornece um caminho bem fundamentado para tais compromissos.

Racionalidade limitada. Herbert Simon (1957) rejeitou a noção de racionalidade perfeita (ou até mesmo aproximadamente perfeita) e a substituiu pela de racionalidade limitada, uma teoria descritiva da tomada de decisões por agentes reais. Ele escreveu:

> A capacidade da mente humana para formular e resolver problemas complexos é muito pequena em comparação com o tamanho dos problemas cuja solução é necessária para se alcançar um comportamento objetivamente racional no mundo real ou mesmo para se obter uma aproximação razoável de tal racionalidade objetiva.

Ele sugeriu que a racionalidade limitada atua principalmente por meio de **satisfação**, ou seja, deliberando durante um tempo longo o bastante para apresentar uma resposta "boa o suficiente". Simon ganhou o Prêmio Nobel de economia por esse trabalho e escreveu muito sobre o tema, descrevendo-o em detalhes (Simon, 1982). Esse parece ser um modelo útil de comportamentos humanos em muitos casos. Entretanto, não é uma especificação formal para agentes inteligentes porque a definição de "boa o suficiente" não é apresentada pela teoria. Além disso, a satisfação parece ser apenas um dos componentes de uma grande variedade de métodos utilizados para lidar com recursos limitados.

Otimização limitada (OL). Um agente com otimização limitada se comporta tão bem quanto possível, *dados seus recursos computacionais*. Isto é, a utilidade esperada do programa de agente para um agente de otimização limitada é pelo menos tão alta quanto a utilidade esperada de qualquer outro programa de agente que funcione na mesma máquina.

Dessas quatro possibilidades, a otimização limitada parece oferecer a melhor esperança para uma forte fundamentação teórica da IA. Ela proporciona a vantagem de ser possível alcançá-la: sempre existe pelo menos um programa que é o melhor de todos — isso é algo que falta à racionalidade perfeita. Os agentes de otimização limitada são verdadeiramente úteis no mundo real, enquanto os agentes cautelosamente racionais em geral não o são, e os agentes de satisfação podem ser úteis ou não, dependendo apenas de seus próprios caprichos.

A abordagem tradicional em IA tem sido começar com a racionalidade cautelosa e depois assumir compromissos para atender a restrições de recursos. Se os problemas impostos pelas restrições forem secundários, é de esperar que o projeto final seja semelhante ao projeto de um agente de OL. Porém, à medida que as restrições de recursos se tornem mais críticas — por exemplo, conforme o

ambiente fique mais complexo —, esperaremos que os dois projetos divirjam. Na teoria de otimização limitada, essas restrições podem ser manipuladas em forma de princípios.

Até agora, pouco se conhece sobre otimização limitada. É possível construir programas de otimização limitada para máquinas muito simples e para tipos de ambientes um pouco restritos (Etzioni, 1989; Russell *et al*., 1993), mas ainda não temos nenhuma ideia de como serão os programas de otimização limitada para grandes computadores de uso geral em ambientes complexos. Se de fato existir uma teoria construtiva de otimização limitada, teremos de esperar que o projeto de programas com otimização limitada não dependa muito fortemente dos detalhes do computador que estiver sendo utilizado. A pesquisa científica se tornaria muito difícil se a adição de alguns kilobytes de memória a uma máquina com vários gigabytes fizesse diferença significativa para o projeto do programa com otimização limitada.Um modo de ter certeza de que isso não poderá acontecer é relaxar um pouco mais os critérios para a otimização limitada. Por analogia com a noção de complexidade assintótica (Apêndice A), podemos definir a **otimização limitada assintótica** (OLA) como a seguir (Russell e Subramanian, 1995). Suponha um programa *P* com otimização limitada para uma máquina *M* em uma classe de ambientes **E**, onde a complexidade de ambientes em **E** é ilimitada. Então, o programa *P’* será OLA para *M* em **E** se puder superar *P* pela execução em uma máquina *kM* que seja *k* vezes mais rápida (ou maior) que *M*. A menos que *k* seja enorme, ficaremos satisfeitos com um programa que seja OLA para um ambiente não trivial em arquitetura não trivial. Haveria pouco interesse em dedicar um enorme esforço à descoberta de programas de OL em lugar de programas de OLA porque, de qualquer modo, o tamanho e a velocidade das máquinas disponíveis tendem a aumentar por um fator constante em um período de tempo fixo.

Podemos arriscar uma suposição de que os programas de OL ou de OLA para computadores poderosos em ambientes complexos não terão necessariamente uma estrutura simples e elegante. Já vimos que a inteligência de uso geral exige alguma capacidade de reflexão e alguma capacidade deliberativa, uma variedade de formas de conhecimento e de tomada de decisões, mecanismo de aprendizagem e compilação para todas essas formas, métodos para controlar o raciocínio e um grande estoque de conhecimento específico de domínios. Um agente de otimização limitada deve se adaptar ao ambiente em que se encontra, de forma que eventualmente sua organização interna reflita otimizações específicas para o ambiente em particular. Isso é só o que se espera, e é semelhante ao modo como evoluíram os carros de corridas limitados pela capacidade dos motores, acabando por gerar projetos extremamente complexos. Suspeitamos que uma ciência de inteligência artificial baseada na otimização limitada envolverá bastante estudo dos processos que permitem a um programa de agente convergir para a otimização limitada e talvez menor concentração nos detalhes dos confusos programas que resultarão.

Em suma, o conceito de otimização limitada foi proposto como tarefa formal para a pesquisa de IA, ao mesmo tempo bem definida e viável. A otimização limitada especifica *programas* ótimos em vez de *ações* ótimas.

Afinal, as ações são geradas por programas, e é sobre os programas que os projetistas têm controle.

27.4 E SE A IA TIVER SUCESSO?

Na obra *Small World* de David Lodge (1984), um romance sobre o mundo acadêmico da crítica literária, o protagonista causa consternação formulando a um grupo de eminentes mas contraditórios teóricos literários a seguinte pergunta: "*E se vocês estiverem certos*?" Nenhum dos teóricos parece ter considerado essa questão antes, talvez porque debater teorias irrefutáveis seja um fim em si mesmo. Confusão semelhante pode ser evocada formulando-se aos pesquisadores de IA a pergunta: "E se você tiver sucesso?"

Como observamos na Seção 26.3, existem questões éticas a considerar. Os computadores inteligentes são mais poderosos, mas esse poder deve ser usado para o bem ou para mal? Aqueles que buscam desenvolver a IA têm a responsabilidade de ver que o impacto de seu trabalho é positivo. A abrangência do impacto dependerá do grau de sucesso da IA. Até mesmo sucessos modestos em IA já alteraram os rumos do pensamento em ciência da computação (Stein, 2002) e o modo como o desenvolvimento de software é praticado. A IA tornou possível novas aplicações, como sistemas de reconhecimento da fala, sistemas de controle de estoque, sistemas de vigilância, robôs e mecanismos de busca.

Podemos esperar que sucessos de nível médio em IA afetem a vida diária de todo tipo de pessoas. Até agora, as redes de comunicação computadorizadas, como telefones celulares e a Internet, tiveram esse tipo de efeito penetrante na sociedade, mas a IA não tem. Podemos imaginar que assistentes pessoais verdadeiramente úteis para uso em escritórios e em casa teriam grande impacto positivo sobre a vida das pessoas, embora possam causar alguma desarticulação econômica em curto prazo. Uma capacidade tecnológica desse nível também poderia ser aplicada ao desenvolvimento de armas autônomas, o que muitos consideram um desenvolvimento indesejável. Alguns dos maiores problemas sociais que enfrentamos hoje — tais como o aproveitamento da informação genômica para o tratamento de doenças, a gestão eficiente dos recursos energéticos e a verificação dos tratados sobre armas nucleares — estão sendo tratados com a ajuda de tecnologias de IA.

Finalmente, parece provável que um sucesso em grande escala na IA — a criação de inteligência, tanto no nível humano quanto em um nível superior — mudaria a vida de uma grande maioria da espécie humana. A natureza de nosso trabalho e nosso papel seriam alterados, bem como nossa visão da inteligência, da consciência e do destino futuro da raça humana. Nesse nível, os sistemas de IA poderiam representar uma ameaça mais direta à autonomia, à liberdade e até mesmo à sobrevivência humana. Por essas razões, não podemos divorciar a pesquisa da IA de suas consequências éticas (veja a Seção 26.3).

O que nos reservará o futuro? Os autores de ficção científica parecem preferir futuros distópicos a futuros utópicos, provavelmente porque eles geram representações mais interessantes. Porém, até agora, a IA parece se alinhar com outras teorias revolucionárias (impressão, sondagem, viagens aéreas, telefonia) cujas repercussões negativas são superadas por seus aspectos positivos.

Concluindo, vemos que a IA fez grande progresso em sua breve história, mas a frase final do estudo de Alan Turing em *Computing Machinery and Intelligence* (1950) é válida hoje:

> Podemos ver apenas uma pequena distância à frente, mas é possível observar que ainda resta muito a fazer.

APÊNDICE **A**

Fundamentos matemáticos

A. 1 ANÁLISE DE COMPLEXIDADE E NOTAÇÃO O( )

Os cientistas de computação frequentemente se encontram diante da tarefa de comparar algoritmos para ver o quanto eles são rápidos ou a quantidade de memória que eles exigem. Existem duas abordagens para desempenhar essa tarefa. A primeira é o **benchmarking** — a execução dos algoritmos em um computador e a medição da velocidade em segundos e do consumo de memória em bytes. Em última instância, é isso o que realmente importa, mas um benchmarking pode ser insatisfatório devido ao fato de ser muito específico: ele mede o desempenho de um programa específico escrito em uma linguagem específica, funcionando em um computador específico, com um compilador específico e com dados de entrada específicos. A partir do único resultado que o benchmarking fornece, talvez seja difícil prever o quanto o algoritmo se comportaria bem em um compilador, computador ou conjunto de dados diferente. A segunda abordagem baseia-se em uma **análise de algoritmos** matemáticos, independentemente da execução específica e da entrada, como discutido a seguir.

A.1.1 Análise assintótica

Examinaremos a abordagem por meio do exemplo a seguir, um programa para calcular a soma de uma sequência de números:

função SOMATÓRIO(*sequência*) **retorna** um número
  soma ← 0
  para *i* = 1 **até** COMPRIMENTO(*sequência*) faça
    soma ← *soma* + *sequência*[*i*]
  retornar *soma*

A primeira etapa da análise consiste em realizar uma abstração sobre a entrada, a fim de encontrar algum parâmetro ou parâmetros que caracterizem o tamanho da entrada. Nesse exemplo, a entrada pode ser caracterizada pelo comprimento da sequência, que chamaremos *n*. A segunda etapa consiste

em realizar uma abstração sobre a implementação, com o objetivo de encontrar alguma medida que reflita o tempo de execução do algoritmo, mas que não esteja ligada a um compilador ou computador específico. No caso do programa SOMATÓRIO, isso poderia ser apenas o número de linhas de código executadas ou talvez pudesse ser mais detalhado, medindo o número de adições, atribuições, referências a vetores e desvios executados pelo algoritmo. De qualquer modo, isso nos dá uma caracterização do número total de passos executados pelo algoritmo como uma função do tamanho da entrada. Chamaremos essa caracterização de $T(n)$. Se contarmos as linhas de código, teremos $T(n) = 2n + 2$ em nosso exemplo.

Se todos os programas fossem tão simples quanto o SOMATÓRIO, a análise de algoritmos seria um campo trivial. Porém, dois problemas a tornam mais complicada. Primeiro, é raro encontrar um parâmetro como n que caracterize completamente o número de passos executados por um algoritmo. Em vez disso, em geral o melhor que podemos fazer é calcular o $T_{\text{pior}}(n)$ do pior caso ou o $T_{\text{médio}}(n)$ do caso médio. Calcular uma média significa que a análise deve pressupor alguma distribuição nas entradas.

O segundo problema é que os algoritmos tendem a resistir à análise exata. Nesse caso, é necessário recuar até uma aproximação. Dizemos que o algoritmo SOMATÓRIO é $O(n)$, o que significa que sua medida é, no máximo, uma constante vezes n, com a possível exceção de alguns valores pequenos de n. Mais formalmente,

$$T(n) \text{ é } O(f(n)) \text{ se } T(n) \leq kf(n) \text{ para algum } k, \text{ para todo } n > n_0.$$

A notação $O(\ )$ nos fornece aquilo que se denomina **análise assintótica**. Podemos afirmar sem dúvida que, à medida que n se aproxima assintoticamente de infinito, um algoritmo $O(n)$ é melhor que um algoritmo $O(n^2)$. Um único valor de benchmarking poderia não substanciar tal afirmação.

A notação $O(\ )$ realiza uma abstração sobre fatores constantes, o que a torna mais fácil de usar, embora menos precisa que a notação $T(\ )$. Por exemplo, um algoritmo $O(n^2)$ sempre será pior que um algoritmo $O(n)$ em longo prazo, mas, se os dois algoritmos forem $T(n^2 + 1)$ e $T(100n + 1000)$, o algoritmo $O(n^2)$ será de fato melhor para $n < 110$.

Apesar dessa desvantagem, a análise assintótica é a ferramenta mais amplamente utilizada para análise de algoritmos. É precisamente porque a análise realiza a abstração sobre o número exato de operações (ignorando o fator constante k) e sobre o conteúdo exato da entrada (considerando apenas seu tamanho n) que a análise se torna matematicamente possível. A notação $O(\ )$ é um bom compromisso entre precisão e facilidade de análise.

A.1.2 Problemas NP e inerentemente difíceis

A análise de algoritmos e a notação $O(\ )$ nos permitem abordar a eficiência de um algoritmo específico. Porém, elas não têm nenhuma relação com o fato de ser ou não possível existir um algoritmo melhor para determinado problema. O campo da **análise de complexidade** analisa problemas em vez de algoritmos. A primeira divisão bruta se dá entre problemas que podem ser resolvidos em tempo polinomial e problemas que não podem ser resolvidos em tempo polinomial,

não importando que algoritmo seja usado. A classe de problemas polinomiais — aqueles que podem ser resolvidos no tempo $O(n^k)$ para algum k constante — é chamada P. Às vezes, esses problemas se denominam problemas "fáceis" porque a classe contém os problemas com tempos de execução semelhantes a $O(\log n)$ e $O(n)$. Porém, ela também contém os problemas com tempo $O(n^{1000})$ e, assim, o adjetivo "fácil" não deve ser considerado de forma muito literal.

Outra classe importante de problemas é a classe NP, de problemas polinomiais não determinísticos. Um problema está nessa classe se existe algum algoritmo que possa pressupor uma solução e depois verificar se o palpite está correto em tempo polinomial. A ideia é que, se você tiver um número arbitrariamente grande de processadores, de forma que possa experimentar todos os palpites ao mesmo tempo ou, se tiver muita sorte e sempre acertar o palpite na primeira vez, os problemas NP se tornarão problemas P. Uma das maiores questões em aberto em ciência da computação é se a classe NP é equivalente à classe P quando não se tem o luxo de um número infinito de processadores ou conjetura onisciente. A maioria dos cientistas da computação está convencida de que P ≠ NP — de que os problemas NP são inerentemente difíceis e não possuem nenhum algoritmo de tempo polinomial, embora isso nunca tenha sido demonstrado.

As pessoas interessadas em decidir se P = NP examinam uma subclasse de NP chamada problemas **NP-completos**. A palavra "completos" é usada nesse caso no sentido de "mais extremo" e, portanto, se refere aos problemas mais difíceis da classe NP. Foi demonstrado que todos os problemas NP-completos estão em P ou nenhum deles está. Isso torna a classe teoricamente interessante, mas a classe também tem interesse prático porque muitos problemas importantes são reconhecidos como NP-completos. Um exemplo é o problema de satisfatibilidade: dada uma sentença de lógica proposicional, existe uma atribuição de valores-verdade para os símbolos de proposições da sentença que a torne verdadeira? A menos que ocorra um milagre e P = NP, não pode haver nenhum algoritmo que resolva *todos* os problemas de satisfabilidade em tempo polinomial. No entanto, a IA está mais interessada em descobrir se existem algoritmos que funcionam com eficiência em problemas *típicos* extraídos de uma distribuição predeterminada; como vimos no Capítulo 7, existem algoritmos como CAMINHADA-SAT que se saem muito bem em muitos problemas.

A classe **co-NP** é o complemento de NP no sentido de que, para todo problema de decisão em NP, existe um problema correspondente em co-NP com as respostas "sim" e "não" invertidas. Sabemos que P é um subconjunto de NP e de co-NP, e acreditamos que haja problemas em co-NP que não estão em P. Os problemas **co-NP-completos** são os problemas mais difíceis em co-NP.

A classe #P é o conjunto de problemas de contagem que correspondem aos problemas de decisão de NP. Os problemas de decisão têm uma resposta sim ou não: existe uma solução para essa fórmula 3-SAT? Os problemas de contagem têm uma resposta inteira: quantas soluções existem para essa fórmula 3-SAT? Em alguns casos, o problema de contagem é muito mais difícil que o problema de decisão. Por exemplo, decidir se um grafo bipartido tem correspondência perfeita é algo que pode ser feito em tempo $O(VE)$ (onde o grafo tem V vértices e E arestas), mas o problema de contagem "quantas correspondências perfeitas tem esse grafo bipartido?" é #P-completo, indicando que é tão difícil quanto qualquer problema em #P e, desse modo, é pelo menos tão difícil quanto qualquer problema NP.

Também estudamos a classe de problemas PSPACE — aqueles que exigem uma quantidade polinomial de espaço, até mesmo em uma máquina não determinística. Acredita-se que os problemas

PSPACE-difíceis sejam piores que os problemas NP-completos, embora seja possível descobrir que NP = PSPACE, da mesma forma que se poderia concluir que P = NP.

A.2 VETORES, MATRIZES E ÁLGEBRA LINEAR

Os matemáticos definem **vetor** como um membro de um espaço vetorial, mas utilizaremos uma definição mais concreta: um vetor é uma sequência ordenada de valores. Por exemplo, em um espaço bidimensional, temos vetores como $\mathbf{x} = \langle 3, 4\rangle$ e $\mathbf{y} = \langle 0, 2\rangle$. Seguimos a convenção habitual de usar caracteres em negrito para representar nomes de vetores, embora alguns autores utilizem setas ou barras sobre os nomes: x ou y. Os elementos de um vetor podem ser acessados com a utilização de subscritos: $\mathbf{z} = \langle z_1, z_2, \ldots, z_n\rangle$. Um ponto confuso: este livro é um trabalho sintético de muitos subcampos, que de diferentes maneiras chamam seus vetores de sequências, listas ou tuplas e, frequentemente, utilizam as notações $\langle 1, 2\rangle$, [1, 2], ou (1, 2).

As duas operações fundamentais sobre vetores são a adição vetorial e a multiplicação escalar. A adição vetorial $\mathbf{x} + \mathbf{y}$ é a soma dos elementos correspondentes dos vetores: $\mathbf{x} + \mathbf{y} = \langle 3 + 0, 4 + 2\rangle = \langle 3, 6\rangle$. A multiplicação escalar multiplica cada elemento por uma constante: $5\mathbf{x} = \langle 5 \times 3, 5 \times 4\rangle = \langle 15, 20\rangle$.

O comprimento de um vetor é denotado por $|\mathbf{x}|$ e é calculado tomando-se a raiz quadrada da soma dos quadrados dos elementos: $|\mathbf{x}| = \sqrt{(3^2 + 4^2)} = 5$. O produto de ponto (também chamado produto escalar) de dois vetores $\mathbf{x} \cdot \mathbf{y}$ é a soma dos produtos dos elementos correspondentes, isto é, $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$ ou, em nosso caso específico, $\mathbf{x} \cdot \mathbf{y} = 3 \times 0 + 4 \times 2 = 8$.

Os vetores frequentemente são interpretados como segmentos de reta orientados (setas) em um espaço euclidiano n dimensional. Então, a adição vetorial é equivalente a conectar o final de um vetor ao início do outro, e o produto pontual $\mathbf{x} \cdot \mathbf{y}$ é igual a $|\mathbf{x}|\,|\mathbf{y}| \cos \theta$, onde θ é o ângulo entre $\mathbf{x}$ e $\mathbf{y}$.

Uma **matriz** é um array retangular de valores organizados em linhas e colunas. Aqui temos uma matriz A de tamanho 3 × 4:

$$\begin{pmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \mathbf{A}_{1,3} & \mathbf{A}_{1,4} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \mathbf{A}_{2,3} & \mathbf{A}_{2,4} \\ \mathbf{A}_{3,1} & \mathbf{A}_{3,2} & \mathbf{A}_{3,3} & \mathbf{A}_{3,4} \end{pmatrix}$$

O primeiro índice de $\mathbf{A}_{i,j}$ especifica a linha e o segundo especifica a coluna. Em linguagem de programação, $\mathbf{A}_{i,j}$ frequentemente é escrito como `A[i,j]` ou `A[i][j]`.

A soma de duas matrizes é definida pela adição de elementos correspondentes; desse modo, $(\mathbf{A}+\mathbf{B})_{i,j} = \mathbf{A}_{i,j} + \mathbf{B}_{i,j}$ (a soma é indefinida se **A** e **B** têm tamanhos diferentes). Também podemos definir a multiplicação de uma matriz por um escalar: $(c\mathbf{A})_{i,j} = c\mathbf{A}_{i,j}$. A multiplicação de matrizes (o produto de duas matrizes) é mais complicada. O produto **AB** é definido apenas se **A** tem o tamanho $a \times b$ e **B** tem o tamanho $b \times c$ (isto é, a segunda matriz tem um número de linhas igual ao número de colunas da primeira matriz); o resultado é uma matriz de tamanho $a \times c$. Se as matrizes tiverem tamanho

apropriado, o resultado será:

$$(\mathbf{AB})_{i,k} = \sum_j \mathbf{A}_{i,j}\mathbf{B}_{j,k}\ .$$

A multiplicação de matrizes não é comutativa, mesmo para matrizes quadradas: $\mathbf{AB} \neq \mathbf{BA}$ em geral. No entanto, é associativa: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$. Observe que o produto escalar pode ser expresso em termos de uma transposição e uma multiplicação de matrizes: $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^\mathrm{T}\mathbf{y}$.

A **matriz identidade I** tem elementos $\mathbf{I}_{i,j}$ iguais a 1 quando $i = j$ e iguais a 0 em caso contrário. Ela tem a propriedade de que $\mathbf{AI} = \mathbf{A}$ para todo **A**. A transposta de **A**, escrita como $\mathbf{A}^\mathrm{T}$ é formada transformando-se as linhas em colunas e vice-versa ou, de modo mais formal, por $\mathbf{A}^\mathrm{T}{}_{i,\,j} = \mathbf{A}_{j,i}$. O **inverso** de uma matriz quadrada **A** é outra matriz quadrada A^{-1} tal que $\mathbf{A}^{-1}\ \mathbf{A} = \mathbf{I}$. Para uma matriz **singular**, o inverso não existe. Para uma matriz não singular, pode ser calculado no tempo $O(n^3)$.

As matrizes são usadas para resolver sistemas de equações lineares no tempo $O(n^3)$; o tempo é dominado pela inversão de uma matriz de coeficientes. Considere o conjunto de equações a seguir, para o qual queremos encontrar uma solução em x, y e z:

$$+2x + y - z = 8$$
$$-3x - y + 2z = -11$$
$$-2x + y + 2x = -3.$$

Podemos representar esse sistema como a equação matricial $\mathbf{A}\ \mathbf{x} = \mathbf{b}$, onde

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 8 \\ -11 \\ -3 \end{pmatrix}.$$

Para resolver $\mathbf{A}\ \mathbf{x} = \mathbf{b}$, multiplicamos ambos os lados por $\mathbf{A}^{-1}$, produzindo $\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b}$, que, simplificando, dá $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. Depois de inverter **A** e multiplicar por **b**, obtemos a resposta

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix}.$$

A.3 DISTRIBUIÇÕES DE PROBABILIDADE

Uma probabilidade é uma medida sobre um conjunto de eventos que satisfaz a três axiomas:

1. A medida de cada evento está entre 0 e 1. Escrevemos essa medida como $0 \leq P(X = x_i) \leq 1$, onde X é uma variável aleatória que representa um evento e x_i são os valores possíveis de X. Em geral, as variáveis aleatórias são denotadas por letras maiúsculas e seus valores são

representados por letras minúsculas.

2. A medida do conjunto inteiro é 1; isto é, $\sum_{i=1}^{n} P(X = x_i) = 1$·
3. A probabilidade de uma união de eventos disjuntos é a soma das probabilidades dos eventos individuais; ou seja, $P(X = x_1 \vee X = x_2) = P(X = x_1) + P(X = x_2)$, onde x_1 e x_2 são disjuntos.

Um **modelo probabilístico** consiste em um espaço amostral de resultados possíveis mutuamente exclusivos, juntamente com uma medida de probabilidade para cada resultado. Por exemplo, em um modelo de previsão do tempo, os resultados poderiam ser ensolarado, nublado, chuvoso e com neve. Um subconjunto desses resultados constitui um evento. Por exemplo, o evento de precipitação é o subconjunto {*chuvoso*, *com neve*}.

Usamos **P**(X) para denotar o vetor de valores $\langle P(X = x_i), \ldots, P(X = x_n) \rangle$. Também empregamos $P(x_i)$ como abreviação para $P(X = x_i)$, e $\sum_x P(x)$ para *Pin* $\sum_{i=1}^{n} P(X = x_i)$·

A probabilidade condicional $P(B \mid A)$ é definida como $P(B \cap A)/P(A)$. Ae B são condicionalmente independentes se $P(B \mid A) = P(B)$ (ou, de modo equivalente, $P(A \mid B) = P(A)$). Para variáveis contínuas, existe um número infinito de valores e, a menos que existam picos de pontos, a probabilidade de qualquer dos valores será 0. Desse modo, definimos uma **função de densidade de probabilidade**, que também denotamos como $P(\cdot)$, mas que tem significado ligeiramente diferente da função de probabilidade discreta. A função de densidade **V.O. 1057** $P(X = x)$ definida como a razão entre a probabilidade de X ficar dentro de um intervalo em torno de x, dividida pela largura do intervalo, à medida que a largura do intervalo tende a zero:

$$P(x) = \lim_{dx \to 0} P(x \leq X \leq x + dx)/dx \ .$$

A função de densidade deve ser não negativa para todo x e deve ter:

$$\int_{-\infty}^{\infty} P(x)\, dx = 1 \ .$$

Também podemos definir uma **função de densidade de probabilidade cumulativa** $F_X(x)$, que é a probabilidade de uma variável aleatória ser menor que x:

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} P(u)\, du \ .$$

Observe que a função de densidade de probabilidade tem unidades, enquanto a função de probabilidade discreta não tem unidades. Por exemplo, se X for medido em segundos, a densidade será medida em Hz (isto é, 1/sec). Se for um ponto em um espaço tridimensional medido em metros, a densidade será medida em $1/m^3$.

Uma das distribuições de probabilidades mais importantes é a **distribuição gaussiana**, também conhecida como **distribuição normal**. Uma distribuição gaussiana com média m e desvio-padrão s (e,

portanto, com variância s2) é definida como

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)} ,$$

onde x é uma variável contínua que varia de $-\infty$ a $+\infty$. Com a média $\mu = 0$ e a variância s2 = 1, conseguimos o caso especial da **distribuição normal padrão**. Para uma distribuição sobre um vetor **x** em n dimensões, existe a distribuição **gaussiana multivariada:**

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{\Sigma}|}} e^{-\frac{1}{2}\left((\mathbf{x}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})\right)} ,$$

onde μ é o vetor média e $\sum$ é a **matriz de covariância** (veja adiante).

Em uma dimensão, também podemos definir a função de **distribuição cumulativa** $F(x)$ como a probabilidade de uma variável aleatória ser menor que x. No caso da distribuição normal padrão, isso é dado por

$$F(x) = \int_{-\infty}^{x} P(z)dz = \frac{1}{2}(1 + \text{erf}(\frac{z-\mu}{\sigma\sqrt{2}})) ,$$

onde erf(x) é a chamada **função de erro**, que não tem nenhuma representação de forma fechada.

O **teorema do limite central** afirma que a média de n variáveis aleatórias tende a uma distribuição normal conforme n tende a infinito. Isso é válido para quase toda coleção de variáveis aleatórias, a menos que a variância de qualquer subconjunto finito de variáveis domine as outras.

O **valor esperado** de uma variável aleatória, $E(X)$, é a média ou valor médio, ponderado pela probabilidade de cada valor. Para uma variável discreta é:

$$E(X) = \sum_i x_i P(X = x_i) .$$

Para uma variável contínua, substitua o somatório por uma integral sobre a função de densidade de probabilidade, $P(x)$:

$$E(X) = \int_{-\infty}^{\infty} xP(x)\,dx ,$$

A **raiz média quadrática**, RMS, de um conjunto de valores (muitas vezes amostras de uma variável aleatória) é a raiz quadrada da média dos quadrados dos valores,

$$RMS(x_1, \ldots, x_n) = \sqrt{\frac{x_1^2 + \ldots + x_n^2}{n}} .$$

A **covariância** de duas variáveis aleatórias é a expectativa do produto de suas diferenças a partir de suas médias:

$$\text{cov}(X, Y) = E((X - \mu_X)(Y - \mu_Y)) .$$

A **matriz de covariância**, muitas vezes indicada por $\sum$ é uma matriz de covariâncias entre elementos de um vetor de variáveis aleatórias. Dado $\mathbf{X} = \langle X_i, \ldots, x_n \rangle^T$, as entradas da matriz de covariância são as seguintes:

$$\mathbf{\Sigma}_{i,j} = \text{cov}(X_i, X_j) = E((X_i - \mu_i)(X_j - \mu_j)) .$$

Alguns pontos diversos adicionais: usamos $\log(x)$ para o logaritmo natural, $\log_e(x)$. Usamos $\text{argmax}_x f(x)$ para o valor de x para o qual $f(x)$ é o máximo.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

A notação $O(\ )$, amplamente utilizada em ciência da computação hoje em dia, foi introduzida primeiro no contexto da teoria dos números pelo matemático alemão P. G. H. Bachmann (1894). O conceito de NP-completeza foi criado por Cook (1971), e o método moderno para estabelecer uma redução de um problema a outro se deve a Karp (1972). Cook e Karp ganharam o prêmio Turing, a mais alta honra em ciência da computação, por seu trabalho.

Os trabalhos clássicos sobre a análise e o projeto de algoritmos incluem os de Knuth (1973) e os de Aho, Hopcroft e Ullman (1974); contribuições mais recentes são as de Tarjan (1983) e as de Cormen, Leiserson e Rivest (1990). Esses livros enfatizam o projeto e a análise de algoritmos para a resolução de problemas tratáveis. Para examinar a teoria da NP-completeza e outras formas de intratabilidade, consulte Garey e Johnson (1979) ou Papadimitriou (1994). Bons textos sobre probabilidade incluem os de Chung (1979), Ross (1988) e Bertsekas e Tsitsiklis (2008).

APÊNDICE **B**

Notas sobre linguagens e algoritmos

B.1 DEFINIÇÃO DE LINGUAGENS COM A FORMA DE BACKUS–NAUR (BNF)

Neste livro, definimos várias linguagens, inclusive as linguagens de lógica proposicional, lógica de primeira ordem, e um subconjunto de linguagem natural. Uma linguagem formal é definida como um conjunto de cadeias, onde cada cadeia é uma sequência de símbolos. Todas as linguagens em que estamos interessados consistem em um conjunto infinito de cadeias e, portanto, precisamos de um modo conciso para caracterizar o conjunto. Fazemos isso utilizando uma **gramática**. O tipo particular de gramática que utilizamos é chamado de **gramática livre de contexto** porque cada expressão tem a mesma forma em qualquer contexto. Escrevemos nossas gramáticas usando um formalismo chamado forma de **Backus−Naur** (**BNF**). Há quatro componentes em uma gramática BNF:

- Um conjunto de **símbolos terminais**. Esses são os símbolos ou palavras que compõem as sequências da linguagem. Eles podem ser as letras (**A, B, C,**…**)** ou as palavras (**a, abade, ábaco**, …), ou os símbolos que forem apropriados para o domínio.
- Um conjunto de **símbolos não terminais** que dividem as subfrases da linguagem em categorias. Por exemplo, o símbolo não terminal *SintagmaNominal* denota um conjunto infinito de cadeias que inclui “você” e “o grande cão feroz”.
- Um **símbolo inicial**, que é o símbolo não terminal que denota as cadeias completas da linguagem. Em linguagem natural, esse símbolo é a *Sentença*; para a aritmética, poderia ser *Expr*, e para linguagens de programação é *Programa.*
- Um conjunto de **regras de reescrita**, com a forma *LE* → *LD*, onde *LE* (lado esquerdo) é um símbolo não terminal e *LD* (lado direito) é uma sequência de zero ou mais símbolos. Eles podem ser tanto símbolos terminais como não terminais, ou o símbolo, que é usado para denotar a cadeia vazia.

Uma regra de uma reescrita da forma

Sentença → *SintagmaNominal SintagmaVerbal*

significa que sempre que temos duas cadeias divididas em categorias como *SintagmaNominal* e *SintagmaVerbal*, podemos reuni-las e definir a categoria resultante como uma *Sentença*. Sendo uma

abreviação, as duas regras ($S \rightarrow A$) e ($S \rightarrow B$) podem ser escritas como ($S \rightarrow A \mid B$).

Aqui está uma gramática BNF para expressões aritméticas simples:

$$
\begin{array}{lll}
\mathit{Expr} & \rightarrow & \mathit{Expr}\ \mathit{Operador}\ \mathit{Expr} \mid (\mathit{Expr}) \mid \mathit{Número} \\
\mathit{Número} & \rightarrow & \mathit{Dígito} \mid \mathit{Número}\ \mathit{Dígito} \\
\mathit{Dígito} & \rightarrow & \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9} \\
\mathit{Operador} & \rightarrow & + \mid - \mid \div \mid \times
\end{array}
$$

Abordamos as linguagens e as gramáticas em mais detalhes no Capítulo 22. É bom lembrar que há notações ligeiramente diferentes para BNF em outros livros; por exemplo, você pode ver <*Dígito*> em vez de *Dígito* para um não terminal, 'palavra' em vez de **palavra** para um terminal ou:: = em vez de → em uma regra.

B.2 DESCRIÇÃO DE ALGORITMOS COM PSEUDOCÓDIGO

Neste livro, os algoritmos estão descritos em pseudocódigo. A maioria dos pseudocódigos deve ser familiar aos usuários de linguagens como Java, C++ ou Lisp. Em alguns lugares usamos fórmulas matemáticas ou linguagem comum para descrever partes que de outra forma seriam mais incômodas. Algumas particularidades devem ser observadas.

- **Variáveis estáticas**: utilizamos a palavra-chave **estática** para afirmar que uma variável recebe um valor inicial na primeira vez em que uma função é chamada e mantém esse valor (ou o valor dado a ela por uma instrução de atribuição subsequente) em todas as chamadas subsequentes à função. Desse modo, as variáveis estáticas são semelhantes às variáveis globais pelo fato de sobreviverem a uma única chamada à sua função, mas só são acessíveis dentro da função. Os programas de agente do livro utilizam variáveis estáticas como *memória.* Programas com variáveis estáticas podem ser implementados como *objetos* em linguagens orientadas a objeto como C++, Java, Python e Smalltalk. Em linguagens funcionais, podem ser implementados por *fechamentos funcionais* em mais de um ambiente que contenha as variáveis necessárias.
- **Funções como valores**: as funções e os procedimentos têm nomes em letras maiúsculas, e as variáveis têm nomes em itálico e letras minúsculas. Assim, na maioria das vezes, uma chamada de função é semelhante à FN(x). No entanto, permitimos que o valor de uma variável seja uma função; por exemplo, se o valor de uma variável f for a função raiz quadrada, então $f(9)$ retornará 3.
- **para cada***:* a notação "**para cada** x **em** c **faça**" significa que o laço será executado com a variável x ligada a elementos sucessivos da coleção c.
- **O recuo é importante:** o recuo é usado para marcar o escopo de um laço ou condicional, como na linguagem Python, ao contrário de Java e C++ (que usam chaves) ou Pascal e Visual Basic (que usam **end**).
- **Atribuição de desestruturação:** a notação "x, $y \leftarrow par$" significa que o lado direito deve ser avaliado como uma tupla de dois elementos, e o primeiro elemento é atribuído a x e o segundo a y. A mesma ideia é usada em "**para cada** x, y **em** *pares* **faça**" e pode ser usada para trocar duas

variáveis: “x, $y \leftarrow y$, x”.

- **Geradores** e **produção**: a notação “gerador G(x) **produz** números” define G como um gerador de função. Isso é mais bem compreendido através de um exemplo. O fragmento de código mostrado na Figura B.1 imprime os números 1, 2, 4,…, e nunca para. A chamada à POTÊNCIA DE-2 retorna um gerador, que por sua vez produz um valor cada vez que o código do laço pede pelo próximo elemento da coleção. Mesmo que a coleção seja infinita, um elemento é enumerado de cada vez.

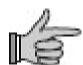
```
gerador POTÊNCIA-DE-2 () retorna ints
   i ← 1
   enquanto verdadeiro faça
      produz i
      i ← 2 × i
--------------------------------------------
para p em potência de 2 () faça
   IMPRIMA (p)
```

Figura B.1 Exemplo de uma função de gerador e sua invocação dentro de um loop.

- **Listas:** [x, y, z] denota uma lista de três elementos. [*primeiro* | *resto*] denota uma lista formada por adição de *primeiro* à lista *resto*. Em Lisp, essa é a função cons.
- **Conjuntos:** $\{x, y, z\}$ denota um conjunto de três elementos. $\{x : p(x)\}$ denota o conjunto de todos os elementos x para os quais $p(x)$ é verdadeiro.
- **Os arrays começam em 1**: salvo disposição em contrário, o primeiro índice de um array é 1, como na notação matemática habitual, e não 0, como em Java e C.

B.3 AJUDA ON-LINE

A maior parte dos algoritmos do livro foi implementada em Java, Lisp, Python e em nosso repositório de código on-line:

 `aima.cs.berkeley.edu`

O mesmo site inclui instruções para o envio de comentários, correções ou sugestões para melhorar o livro e para juntar listas de discussão.

Bibliografia

Aarup, M., Arentoft, M. M., Parrod, Y., Stader, J., and Stokes, I. (1994). OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. In Fox, M. and Zweben, M. (Eds.), *Knowledge Based Scheduling*. Morgan Kaufmann.

Abney, S. (2007). *Semisupervised Learning for Computational Linguistics*. CRC Press.

Abramson, B. and Yung, M. (1989). Divide and conquer under global constraints: A solution to the N-queens problem. *J. Parallel and Distributed Computing*, *6*(3), 649–662.

Achlioptas, D. (2009). Random satisfiability. In Biere, A., Heule, M., van Maaren, H., and Walsh, T. (Eds.), *Handbook of Satisfiability*. IOS Press.

Achlioptas, D., Beame, P., and Molloy, M. (2004). Exponential bounds for DPLL below the satisfiability threshold. In *SODA-04*.

Achlioptas, D., Naor, A., and Peres, Y. (2007). On the maximum satisfiability of random formulas. *JACM*, *54*(2).

Achlioptas, D. and Peres, Y. (2004). The threshold for random k-SAT is 2klog 2 − o(k). *J. American Mathematical Society*, *17*(4), 947–973.

Ackley, D. H. and Littman, M. L. (1991). Interactions between learning and evolution. In Lang-ton, C., Taylor, C., Farmer, J. D., and Ramussen, S. (Eds.), *Artificial Life II*, pp. 487–509. Addison-Wesley.

Adelson-Velsky, G. M., Arlazarov, V. L., Bitman, A. R., Zhivotovsky, A. A., and Uskov, A. V. (1970). Programming a computer to play chess. *Russian Mathematical Surveys*, *25*, 221–262.

Adida, B. and Birbeck, M. (2008). RDFa primer. Tech. rep., W3C.

Agerbeck, C. and Hansen, M. O. (2008). A multi-agent approach to solving *NP*-complete problems. Master's thesis, Technical Univ. of Denmark.

Aggarwal, G., Goel, A., and Motwani, R. (2006). Truthful auctions for pricing search keywords. In *EC-06*, pp. 1–7.

Agichtein, E. and Gravano, L. (2003). Querying text databases for efficient information extraction. In *Proc. IEEE Conference on Data Engineering*.

Agmon, S. (1954). The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, *6*(3), 382–392.

Agre, P. E. and Chapman, D. (1987). Pengi: an implementation of a theory of activity. In *IJCAI-87*, pp. 268–272.

Aho, A. V., Hopcroft, J., and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, *25*, 821–837.

Al-Chang, M., Bresina, J., Charest, L., Chase, A., Hsu, J., Jonsson, A., Kanefsky, B., Morris, P., Rajan, K., Yglesias, J., Chafin, B., Dias, W., and Maldague, P. (2004). MAPGEN: Mixed-Initiative planning and scheduling for the Mars Exploration Rover mission. *IEEE Intelligent Systems*, *19*(1), 8–12.

Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *J. Dynamic Systems, Measurement, and Control*, *97*, 270–277.

Aldous, D. and Vazirani, U. (1994). "Go with the winners" algorithms. In *FOCS-94*, pp. 492–501.

Alekhnovich, M., Hirsch, E. A., and Itsykson, D. (2005). Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. *JAR*, *35*(1–3), 51–72.

Allais, M. (1953). Le comportment de l'homme rationnel devant la risque: critique des postulats et axiomes de l'´ecole Am´ericaine. *Econometrica*, *21*, 503–546.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *CACM*, *26*(11), 832–843.

Allen, J. F. (1984). Towards a general theory of action and time. *AIJ*, *23*, 123–154.

Allen, J. F. (1991). Time and time again: The many ways to represent time. *Int. J. Intelligent Systems*, *6*, 341–355.

Allen, J. F., Hendler, J., and Tate, A. (Eds.). (1990). *Readings in Planning*. Morgan Kaufmann.

Allis, L. (1988). A knowledge-based approach to connect four. The game is solved: White wins. Master's thesis, Vrije Univ., Amsterdam.

Almuallim, H. and Dietterich, T. (1991). Learning with many irrelevant features. In *AAAI-91*, Vol. 2, pp. 547–552.

ALPAC (1966). Language and machines: Computers in translation and linguistics. Tech. rep. 1416, The Automatic Language Processing Advisory Committee of the National Academy of Sciences.

Alterman, R. (1988). Adaptive planning. *Cognitive Science*, *12*, 393–422.

Amarel, S. (1967). An approach to heuristic problem-solving and theorem proving in the propositional calculus. In Hart, J. and Takasu, S. (Eds.), *Systems and Computer Science*. University of Toronto Press.

Amarel, S. (1968). On representations of problems of reasoning about actions. In Michie, D. (Ed.), *Machine Intelligence 3*, Vol. 3, pp. 131–171. Elsevier/North-Holland.

Amir, E. and Russell, S. J. (2003). Logical filtering. In *IJCAI-03*.

Amit, D., Gutfreund, H., and Sompolinsky, H. (1985). Spin-glass models of neural networks. *Physical Review*, *A 32*, 1007–1018.

Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen, F. (1989). HUGIN—A shell for building Bayesian belief universes for expert systems. In *IJCAI-89*, Vol. 2, pp. 1080–1085.

Anderson, J. R. (1980). *Cognitive Psychology and Its Implications*. W. H. Freeman.

Anderson, J. R. (1983). *The Architecture of Cognition*. Harvard University Press.

Andoni, A. and Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS-06*.

Andre, D. and Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *AAAI-02*, pp. 119–125.

Anthony, M. and Bartlett, P. (1999). *Neural Network Learning: Theoretical Foundations*. Cambridge University Press.

Aoki, M. (1965). Optimal control of partially observable Markov systems. *J. Franklin Institute*, *280*(5), 367–386.

Appel, K. and Haken, W. (1977). Every planar map is four colorable: Part I: Discharging. *Illinois J. Math.*, *21*, 429–490.

Appelt, D. (1999). Introduction to information extraction. *CACM*, *12*(3), 161–172.

Apt, K. R. (1999). The essence of constraint propagation. *Theoretical Computer Science*, *221*(1–2), 179–210.

Apt, K. R. (2003). *Principles of Constraint Programming*. Cambridge University Press.

Apte´, C., Damerau, F., and Weiss, S. (1994). Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, *12*, 233–251.

Arbuthnot, J. (1692). *Of the Laws of Chance*. Motte, London. Translation into English, with additions, of Huygens (1657).

Archibald, C., Altman, A., and Shoham, Y. (2009). Analysis of a winning computational billiards player. In *IJCAI-09*.

Ariely, D. (2009). *Predictably Irrational* (Revised edition). Harper.

Arkin, R. (1998). *Behavior-Based Robotics*. MIT Press.

Armando, A., Carbone, R., Compagna, L., Cuellar, J., and Tobarra, L. (2008). Formal analysis of SAML 2.0 web browser single sign-on: Breaking the SAML-based single sign-on for google apps. In *FMSE '08: Proc. 6th ACM workshop on Formal methods in security engineering*, pp. 1–10.

Arnauld, A. (1662). *La logique, ou l'art de penser*. Chez Charles Savreux, au pied de la Tour de Nostre Dame, Paris.

Arora, S. (1998). Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *JACM*, *45*(5), 753–782.

Arunachalam, R. and Sadeh, N. M. (2005). The supply chain trading agent competition. *Electronic Commerce Research and Applications*, *Spring*, 66–84.

Ashby, W. R. (1940). Adaptiveness and equilibrium. *J. Mental Science*, *86*, 478–483.

Ashby, W. R. (1948). Design for a brain. *Electronic Engineering*, *December*, 379–383.

Ashby, W. R. (1952). *Design for a Brain*. Wiley.

Asimov, I. (1942). Runaround. *Astounding Science Fiction*, *March*.

Asimov, I. (1950). *I, Robot*. Doubleday.

Astrom, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, *10*, 174–205.

Audi, R. (Ed.). (1999). *The Cambridge Dictionary of Philosophy*. Cambridge University Press.

Axelrod, R. (1985). *The Evolution of Cooperation*. Basic Books.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. (2007). *The Description Logic Handbook* (2nd edition). Cambridge University Press.

Baader, F. and Snyder, W. (2001). Unification theory. In Robinson, J. and Voronkov, A. (Eds.), *Handbook of Automated Reasoning*, pp. 447–533. Else-vier.

Bacchus, F. (1990). *Representing and Reasoning with Probabilistic Knowledge*. MIT Press.

Bacchus, F. and Grove, A. (1995). Graphical models for preference and utility. In *UAI-95*, pp. 3–10.

Bacchus, F. and Grove, A. (1996). Utility independence in a qualitative decision theory. In *KR-96*, pp. 542–552.

Bacchus, F., Grove, A., Halpern, J. Y., and Koller, D. (1992). From statistics to beliefs. In *AAAI-92*, pp. 602–608.

Bacchus, F. and van Beek, P. (1998). On the conversion between non-binary and binary constraint satisfaction problems. In *AAAI-98*, pp. 311–318.

Bacchus, F. and van Run, P. (1995). Dynamic variable ordering in CSPs. In *CP-95*, pp. 258–275.

Bachmann, P. G. H. (1894). *Die analytische Zahlentheorie*. B. G. Teubner, Leipzig.

Backus, J. W. (1996). Transcript of question and answer session. In Wexelblat, R. L. (Ed.), *History of Programming Languages*, p. 162. Academic Press.

Bagnell, J. A. and Schneider, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *ICRA-01*.

Baker, J. (1975). The Dragon system—An overview. *IEEE Transactions on Acoustics; Speech; and Signal Processing*, *23*, 24–29.

Baker, J. (1979). Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pp. 547–550.

Baldi, P., Chauvin, Y., Hunkapiller, T., and McClure, M. (1994). Hidden Markov models of biological primary sequence information. *PNAS*, *91*(3), 1059–1063.

Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist*, *30*, 441–451. Continued on pages 536–553.

Ballard, B. W. (1983). The *-minimax search procedure for trees containing chance nodes. *AIJ*,

21(3), 327–350.

Baluja, S. (1997). Genetic algorithms and explicit search statistics. In Mozer, M. C., Jordan, M. I., and Petsche, T. (Eds.), *NIPS 9*, pp. 319–325. MIT Press.

Bancilhon, F., Maier, D., Sagiv, Y., and Ullman, J. D. (1986). Magic sets and other strange ways to implement logic programs. In *PODS-86*, pp. 1–16.

Banko, M. and Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In *ACL-01*, pp. 26–33.

Banko, M., Brill, E., Dumais, S. T., and Lin, J. (2002). Askmsr: Question answering using the worldwide web. In *Proc. AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*, pp. 7–9.

Banko, M., Cafarella, M. J., Soderland, S., Broad-head, M., and Etzioni, O. (2007). Open information extraction from the web. In *IJCAI-07*.

Banko, M. and Etzioni, O. (2008). The tradeoffs between open and traditional relation extraction. In *ACL-08*, pp. 28–36.

Bar-Hillel, Y. (1954). Indexical expressions. *Mind*, *63*, 359–379.

Bar-Hillel, Y. (1960). The present status of automatic translation of languages. In Alt, F. L. (Ed.), *Advances in Computers*, Vol. 1, pp. 91–163. Academic Press.

Bar-Shalom, Y. (Ed.). (1992). *Multitargetmultisensor tracking: Advanced applications*. Artech House.

Bar-Shalom, Y. and Fortmann, T. E. (1988). *Tracking and Data Association*. Academic Press.

Bartak, R. (2001). Theory and practice of constraint propagation. In *Proc. Third Workshop on Constraint Programming for Decision and Control (CPDC-01)*, pp. 7–14.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *AIJ*, *73*(1), 81–138.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, *13*, 834–846.

Barto, A. G., Sutton, R. S., and Brouwer, P. S. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, *40*(3), 201–211.

Barwise, J. and Etchemendy, J. (1993). *The Language of First-Order Logic: Including the Macintosh Program Tarski's World 4.0* (Third Revised and Expanded edition). Center for the Study of Language and Information (CSLI).

Barwise, J. and Etchemendy, J. (2002). *Language, Proof and Logic*. CSLI (Univ. of Chicago Press).

Baum, E., Boneh, D., and Garrett, C. (1995). On genetic algorithms. In *COLT-95*, pp. 230–239.

Baum, E. and Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, *1*(1), 151–160.

Baum, E. and Smith, W. D. (1997). A Bayesian approach to relevance in game playing. *AIJ*, *97*(1–2), 195–242.

Baum, E. and Wilczek, F. (1988). Supervised learning of probability distributions by neural networks. In Anderson, D. Z. (Ed.), *Neural Information Processing Systems*, pp. 52–61. American Institute of Physics.

Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, *41*.

Baxter, J. and Bartlett, P. (2000). Reinforcement learning in POMDP's via direct gradient ascent. In *ICML-00*, pp. 41–48.

Bayardo, R. J. and Miranker, D. P. (1994). An optimal backtrack algorithm for tree-structured constraint satisfaction problems. *AIJ*, *71*(1), 159–181.

Bayardo, R. J. and Schrag, R. C. (1997). Using CSP look-back techniques to solve real-world SAT instances. In *AAAI-97*, pp. 203–208.

Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, *53*, 370–418.

Beal, D. F. (1980). An analysis of minimax. In Clarke, M. R. B. (Ed.), *Advances in Computer Chess 2*, pp. 103–109. Edinburgh University Press.

Beal, J. and Winston, P. H. (2009). The new frontier of human-level artificial intelligence. *IEEE Intelligent Systems*, *24*(4), 21–23.

Beckert, B. and Posegga, J. (1995). Leantap: Lean, tableau-based deduction. *JAR*, *15*(3), 339–358.

Beeri, C., Fagin, R., Maier, D., and Yannakakis, M. (1983). On the desirability of acyclic database schemes. *JACM*, *30*(3), 479–513.

Bekey, G. (2008). *Robotics: State Of The Art And Future Challenges*. Imperial College Press.

Bell, C. and Tate, A. (1985). Using temporal constraints to restrict search in a planner. In *Proc. Third Alvey IKBS SIG Workshop*.

Bell, J. L. and Machover, M. (1977). *A Course in Mathematical Logic*. Elsevier/North-Holland.

Bellman, R. E. (1952). On the theory of dynamic programming. *PNAS*, *38*, 716–719.

Bellman, R. E. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton University Press.

Bellman, R. E. (1965). On the application of dynamic programming to the determination of optimal play in chess and checkers. *PNAS*, *53*, 244–246.

Bellman, R. E. (1978). *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company.

Bellman, R. E. (1984). *Eye of the Hurricane*. World Scientific.

Bellman, R. E. and Dreyfus, S. E. (1962). *Applied Dynamic Programming*. Princeton University Press.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Belongie, S., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *PAMI*, *24*(4), 509–522.

Ben-Tal, A. and Nemirovski, A. (2001). *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. SIAM (Society for Industrial and Applied Mathematics).

Bender, E. A. (1996). *Mathematical methods in artificial intelligence*. IEEE Computer Society Press.

Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J. (Eds.), *Large-Scale Kernel Machines*. MIT Press.

Bentham, J. (1823). *Principles of Morals and Legislation*. Oxford University Press, Oxford, UK. Original work published in 1789.

Berger, J. O. (1985). *Statistical Decision Theory and Bayesian Analysis*. Springer Verlag.

Berkson, J. (1944). Application of the logistic function to bio-assay. *JASA*, *39*, 357–365.

Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982). *Winning Ways, For Your Mathematical Plays*. Academic Press.

Berlekamp, E. R. and Wolfe, D. (1994). *Mathematical Go: Chilling Gets the Last Point*. A.K. Peters.

Berleur, J. and Brunnstein, K. (2001). *Ethics of Computing: Codes, Spaces for Discussion and Law*. Chapman and Hall.

Berliner, H. J. (1979). The B* tree search algorithm: A best-first proof procedure. *AIJ*, *12*(1), 23–40.

Berliner, H. J. (1980a). Backgammon computer program beats world champion. *AIJ*, *14*, 205–220.

Berliner, H. J. (1980b). Computer backgammon. *Scientific American*, *249*(6), 64–72.

Bernardo, J. M. and Smith, A. F. M. (1994). *Bayesian Theory*. Wiley.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, *284*(5), 34–43.

Bernoulli, D. (1738). Specimen theoriae novae de mensura sortis. *Proc. St. Petersburg Imperial Academy of Sciences*, *5*, 175–192.

Bernstein, A. and Roberts, M. (1958). Computer vs. chess player. *Scientific American*, *198*(6), 96–105.

Bernstein, P. L. (1996). *Against the Odds: The Remarkable Story of Risk*. Wiley.

Berrou, C., Glavieux, A., and Thitimajshima, P. (1993). Near Shannon limit error control-correcting coding and decoding: Turbo-codes. 1. In *Proc. IEEE International Conference on Communications*, pp. 1064–1070.

Berry, D. A. and Fristedt, B. (1985). *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall.

Bertele, U. and Brioschi, F. (1972). *Nonserial dynamic programming*. Academic Press.

Bertoli, P., Cimatti, A., and Roveri, M. (2001a). Heuristic search + symbolic model checking = efficient conformant planning. In *IJCAI-01*, pp. 467–472.

Bertoli, P., Cimatti, A., Roveri, M., and Traverso, P. (2001b). Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI-01*, pp. 473–478.

Bertot, Y., Casteran, P., Huet, G., and Paulin-Mohring, C. (2004). *Interactive Theorem Proving and Program Development*. Springer.

Bertsekas, D. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall.

Bertsekas, D. and Tsitsiklis, J. N. (1996). *Neurodynamic programming*. Athena Scientific.

Bertsekas, D. and Tsitsiklis, J. N. (2008). *Introduction to Probability* (2nd edition). Athena Scientific.

Bertsekas, D. and Shreve, S. E. (2007). *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific.

Bessiere`, C. (2006). Constraint propagation. In Rossi, F., van Beek, P., and Walsh, T. (Eds.), *Handbook of Constraint Programming*. Elsevier.

Bhar, R. and Hamori, S. (2004). *Hidden Markov Models: Applications to Financial Economics*. Springer.

Bibel, W. (1993). *Deduction: Automated Logic*. Academic Press.

Biere, A., Heule, M., van Maaren, H., and Walsh, T. (Eds.). (2009). *Handbook of Satisfiability*. IOS Press.

Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T., and Szafron, D. (2003). Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI-03*.

Binder, J., Koller, D., Russell, S. J., and Kanazawa, K. (1997a). Adaptive probabilistic networks with hidden variables. *Machine Learning*, *29*, 213–244.

Binder, J., Murphy, K., and Russell, S. J. (1997b). Space-efficient inference in dynamic probabilistic networks. In *IJCAI-97*, pp. 1292–1296.

Binford, T. O. (1971). Visual perception by computer. Invited paper presented at the IEEE Systems Science and Cybernetics Conference, Miami.

Binmore, K. (1982). *Essays on Foundations of Game Theory*. Pitman.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. Springer-Verlag.

Bisson, T. (1990). They're made out of meat. *Omni Magazine*.

Bistarelli, S., Montanari, U., and Rossi, F. (1997). Semiring-based constraint satisfaction and optimization. *JACM*, *44*(2), 201–236.

Bitner, J. R. and Reingold, E. M. (1975). Backtrack programming techniques. *CACM*, *18*(11), 651–656.

Bizer, C., Auer, S., Kobilarov, G., Lehmann, J., and Cyganiak, R. (2007). DBPedia – querying wikipedia like a database. In *Developers Track Presentation at the 16th International Conference on World Wide Web*.

Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., and Weglarz, J. (2007). *Handbook on Scheduling: Models and Methods for Advanced Planning (International Handbooks on Information Systems)*. Springer-Verlag New York, Inc.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2001). Latent Dirichlet Allocation. In *Neural Information Processing Systems*, Vol. 14.

Blinder, A. S. (1983). Issues in the coordination of monetary and fiscal policies. In *Monetary Policy Issues in the 1980s*. Federal Reserve Bank, Kansas City, Missouri.

Bliss, C. I. (1934). The method of probits. *Science*, *79*(2037), 38–39.

Block, H. D., Knight, B., and Rosenblatt, F. (1962). Analysis of a four-layer series-coupled perceptron. *Rev. Modern Physics*, *34*(1), 275–282.

Blum, A. L. and Furst, M. (1995). Fast planning through planning graph analysis. In *IJCAI-95*, pp. 1636–1642.

Blum, A. L. and Furst, M. (1997). Fast planning through planning graph analysis. *AIJ*, *90*(1–2), 281–300.

Blum, A. L. (1996). On-line algorithms in machine learning. In *Proc. Workshop on On-Line Algorithms, Dagstuhl*, pp. 306–325.

Blum, A. L. and Mitchell, T. M. (1998). Combining labeled and unlabeled data with co-training. In *COLT-98*, pp. 92–100.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *JACM*, *36*(4), 929–965.

Bobrow, D. G. (1967). Natural language input for a computer problem solving system. In Minsky, M. L. (Ed.), *Semantic Information Processing*, pp. 133–215. MIT Press.

Bobrow, D. G., Kaplan, R., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS, a frame driven dialog system. *AIJ*, *8*, 155–173.

Boden, M. A. (1977). *Artificial Intelligence and Natural Man*. Basic Books.

Boden, M. A. (Ed.). (1990). *The Philosophy of Artificial Intelligence*. Oxford University Press.

Bolognesi, A. and Ciancarini, P. (2003). Computer programming of kriegspiel endings: The case of KR vs. k. In *Advances in Computer Games 10*.

Bonet, B. (2002). An epsilon-optimal grid-based algorithm for partially observable Markov decision processes. In *ICML-02*, pp. 51–58.

Bonet, B. and Geffner, H. (1999). Planning as heuristic search: New results. In *ECP-99*, pp. 360–

372.

Bonet, B. and Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In *ICAPS-00*, pp. 52–61.

Bonet, B. and Geffner, H. (2005). An algorithm better than AO* ? In *AAAI-05*.

Boole, G. (1847). *The Mathematical Analysis of Logic: Being an Essay towards a Calculus of Deductive Reasoning*. Macmillan, Barclay, and Macmillan, Cambridge.

Booth, T. L. (1969). Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, pp. 74–81.

Borel, E. (1921). La théorie du jeu et les équations intégrales a noyau symétrique. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, *173*, 1304–1308.

Borenstein, J., Everett, B., and Feng, L. (1996). *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd.

Borenstein, J. and Koren., Y. (1991). The vector field histogram—Fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, *7*(3), 278–288.

Borgida, A., Brachman, R. J., McGuinness, D., and Alperin Resnick, L. (1989). CLASSIC: A structural data model for objects. *SIGMOD Record*, *18*(2), 58–67.

Boroditsky, L. (2003). Linguistic relativity. In Nadel, L. (Ed.), *Encyclopedia of Cognitive Science*, pp. 917–921. Macmillan.

Boser, B., Guyon, I., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *COLT-92*.

Bosse, M., Newman, P., Leonard, J., Soika, M., Feiten, W., and Teller, S. (2004). Simultaneous localization and map building in large-scale cyclic environments using the atlas framework. *Int. J. Robotics Research*, *23*(12), 1113–1139.

Bourzutschky, M. (2006). 7-man endgames with pawns. *CCRL Discussion Board*, *kirill-kryukov.com/chess/discussion-board/viewtopic.php?t=805*.

Boutilier, C. and Brafman, R. I. (2001). Partial-order planning with concurrent interacting actions. *JAIR*, *14*, 105–136.

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *AIJ*, *121*, 49–107.

Boutilier, C., Reiter, R., and Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *IJCAI-01*, pp. 467–472.

Boutilier, C., Friedman, N., Goldszmidt, M., and Koller, D. (1996). Context-specific independence in Bayesian networks. In *UAI-96*, pp. 115–123.

Bouzy, B. and Cazenave, T. (2001). Computer go: An AI oriented survey. *AIJ*, *132*(1), 39–103.

Bowerman, M. and Levinson, S. (2001). *Language acquisition and conceptual development*. Cambridge University Press.

Bowling, M., Johanson, M., Burch, N., and Szafron, D. (2008). Strategy evaluation in extensive games with importance sampling. In *ICML-08*.

Box, G. E. P. (1957). Evolutionary operation: A method of increasing industrial productivity. *Applied Statistics*, *6*, 81–101.

Box, G. E. P., Jenkins, G., and Reinsel, G. (1994). *Time Series Analysis: Forecasting and Control* (3rd edition). Prentice Hall.

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, *49*(2–3), 233–246.

Boyan, J. A. and Moore, A. W. (1998). Learning evaluation functions for global optimization and Boolean satisfiability. In *AAAI-98*.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Boyen, X., Friedman, N., and Koller, D. (1999). Discovering the hidden structure of complex dynamic systems. In *UAI-99*.

Boyer, R. S. and Moore, J. S. (1979). *A Computational Logic*. Academic Press.

Boyer, R. S. and Moore, J. S. (1984). Proof checking the RSA public key encryption algorithm. *American Mathematical Monthly*, *91*(3), 181–189.

Brachman, R. J. (1979). On the epistemological status of semantic networks. In Findler, N. V. (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, pp. 3–50. Academic Press.

Brachman, R. J., Fikes, R. E., and Levesque, H. J. (1983). Krypton: A functional approach to knowledge representation. *Computer*, *16*(10), 67–73.

Brachman, R. J. and Levesque, H. J. (Eds.). (1985). *Readings in Knowledge Representation*. Morgan Kaufmann.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, *22*, 33–57.

Brafman, O. and Brafman, R. (2009). *Sway: The Irresistible Pull of Irrational Behavior*. Broadway Business.

Brafman, R. I. and Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS-08*, pp. 28–35.

Brafman, R. I. and Tennenholtz, M. (2000). A near optimal polynomial time algorithm for learning in certain classes of stochastic games. *AIJ*, *121*, 31–47.

Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. MIT Press.

Bransford, J. and Johnson, M. (1973). Consideration of some problems in comprehension. In Chase, W. G. (Ed.), *Visual Information Processing*. Academic Press.

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large language models in machine translation. In *EMNLP-CoNLL-2007: Proc. 2007 Joint Conference on Empirical Methods in*

Natural Language Processing and Computational Natural Language Learning, pp. 858–867.

Bratko, I. (1986). *Prolog Programming for Artificial Intelligence* (1st edition). Addison-Wesley.

Bratko, I. (2001). *Prolog Programming for Artificial Intelligence* (Third edition). Addison-Wesley.

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*. Harvard University Press.

Bratman, M. E. (1992). Planning and the stability of intention. *Minds and Machines*, *2*(1), 1–16.

Breese, J. S. (1992). Construction of belief and decision networks. *Computational Intelligence*, *8*(4), 624–647.

Breese, J. S. and Heckerman, D. (1996). Decision-theoretic troubleshooting: A framework for repair and experiment. In *UAI-96*, pp. 124–132.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Breiman, L., Friedman, J., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group.

Brelaz, D. (1979). New methods to color the vertices of a graph. *CACM*, *22*(4), 251–256.

Brent, R. P. (1973). *Algorithms for minimization without derivatives*. Prentice-Hall.

Bresnan, J. (1982). *The Mental Representation of Grammatical Relations*. MIT Press.

Brewka, G., Dix, J., and Konolige, K. (1997). *Nononotonic Reasoning: An Overview*. CSLI Publications.

Brickley, D. and Guha, R. V. (2004). RDF vocabulary description language 1.0: RDF schema. Tech. rep., W3C.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Fogelman Souli´e, F. and H´erault, J. (Eds.), *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag.

Briggs, R. (1985). Knowledge representation in Sanskrit and artificial intelligence. *AIMag*, *6*(1), 32–39.

Brin, D. (1998). *The Transparent Society*. Perseus.

Brin, S. (1999). Extracting patterns and relations from the world wide web. Technical report 1999-65, Stanford InfoLab.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In *Proc. Seventh World Wide Web Conference*.

Bringsjord, S. (2008). If I were judge. In Epstein, R., Roberts, G., and Beber, G. (Eds.), *Parsing the Turing Test*. Springer.

Broadbent, D. E. (1958). *Perception and Communication*. Pergamon.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *2*, 14–23.

Brooks, R. A. (1989). Engineering approach to building complete, intelligent beings. *Proc. SPIE—the International Society for Optical Engineering*, *1002*, 618–625.

Brooks, R. A. (1991). Intelligence without representation. *AIJ*, *47*(1–3), 139–159.

Brooks, R. A. and Lozano-Perez, T. (1985). A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man and Cybernetics*, *15*(2), 224–233.

Brown, C., Finkelstein, L., and Purdom, P. (1988). Backtrack searching in the presence of symmetry. In Mora, T. (Ed.), *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pp. 99–110. Springer-Verlag.

Brown, K. C. (1974). A note on the apparent bias of net revenue estimates. *J. Finance*, *29*, 1215–1216.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Mercer, R. L., and Roossin, P. (1988). A statistical approach to language translation. In *COLING-88*, pp. 71–76.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, *19*(2), 263–311.

Brownston, L., Farrell, R., Kant, E., and Martin, N. (1985). *Programming expert systems in OPS5: An introduction to rule-based programming*. Addison-Wesley.

Bruce, V., Georgeson, M., and Green, P. (2003). *Visual Perception: Physiology, Psychology and Ecology*. Psychology Press.

Bruner, J. S., Goodnow, J. J., and Austin, G. A. (1957). *A Study of Thinking*. Wiley.

Bryant, B. D. and Miikkulainen, R. (2007). Acquiring visibly intelligent behavior with example-guided neuroevolution. In *AAAI-07*.

Bryce, D. and Kambhampati, S. (2007). A tutorial on planning graph-based reachability heuristics. *AIMag*, *Spring*, 47–83.

Bryce, D., Kambhampati, S., and Smith, D. E. (2006). Planning graph heuristics for belief space search. *JAIR*, *26*, 35–99.

Bryson, A. E. and Ho, Y.-C. (1969). *Applied Optimal Control*. Blaisdell.

Buchanan, B. G. and Mitchell, T. M. (1978). Model-directed learning of production rules. In Waterman, D. A. and Hayes-Roth, F. (Eds.), *Pattern-Directed Inference Systems*, pp. 297–312. Academic Press.

Buchanan, B. G., Mitchell, T. M., Smith, R. G., and Johnson, C. R. (1978). Models of learning systems. In *Encyclopedia of Computer Science and Technology*, Vol. 11. Dekker.

Buchanan, B. G. and Shortliffe, E. H. (Eds.). (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.

Buchanan, B. G., Sutherland, G. L., and Feigenbaum, E. A. (1969). Heuristic DENDRAL: A program for generating explanatory hypotheses in organic chemistry. In Meltzer, B., Michie, D., and Swann, M. (Eds.), *Machine Intelligence 4*, pp. 209–254. Edinburgh University Press.

Buehler, M., Iagnemma, K., and Singh, S. (Eds.). (2006). *The 2005 DARPA Grand Challenge: The Great Robot Race*. Springer-Verlag.

Bunt, H. C. (1985). The formal representation of (quasi-) continuous concepts. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 2, pp. 37–70. Ablex.

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *AIJ*, *114*(1–2), 3–55.

Buro, M. (1995). ProbCut: An effective selective extension of the alpha-beta algorithm. *J. International Computer Chess Association*, *18*(2), 71–76.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *AIJ*, *134*(1–2), 85–99.

Burstein, J., Leacock, C., and Swartz, R. (2001). Automated evaluation of essays and short answers. In *Fifth International Computer Assisted Assessment (CAA) Conference*.

Burton, R. (2009). *On Being Certain: Believing You Are Right Even When You're Not*. St. Martin's Griffin.

Buss, D. M. (2005). *Handbook of evolutionary psychology*. Wiley.

Butler, S. (1863). Darwin among the machines. *The Press (Christchurch, New Zealand)*, *June 13*.

Bylander, T. (1992). Complexity results for serial decomposability. In *AAAI-92*, pp. 729–734.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *AIJ*, *69*, 165–204.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, *16*(5), 1190–1208.

Cabeza, R. and Nyberg, L. (2001). Imaging cognition II: An empirical review of 275 PET and fMRI studies. *J. Cognitive Neuroscience*, *12*, 1–47.

Cafarella, M. J., Halevy, A., Zhang, Y., Wang, D. Z., and Wu, E. (2008). Webtables: Exploring the power of tables on the web. In *VLDB-2008*.

Calvanese, D., Lenzerini, M., and Nardi, D. (1999). Unifying class-based representation formalisms. *JAIR*, *11*, 199–240.

Campbell, M. S., Hoane, A. J., and Hsu, F.-H. (2002). Deep Blue. *AIJ*, *134*(1–2), 57–83.

Canny, J. and Reif, J. (1987). New lower bound techniques for robot motion planning problems. In *FOCS-87*, pp. 39–48.

Canny, J. (1986). A computational approach to edge detection. *PAMI*, *8*, 679–698.

Canny, J. (1988). *The Complexity of Robot Motion Planning*. MIT Press.

Capen, E., Clapp, R., and Campbell, W. (1971). Competitive bidding in high-risk situations. *J. Petroleum Technology*, *23*, 641–653.

Caprara, A., Fischetti, M., and Toth, P. (1995). A heuristic method for the set covering problem. *Operations Research*, *47*, 730–743.

Carbonell, J. G. (1983). Derivational analogy and its role in problem solving. In *AAAI-83*, pp. 64–69.

Carbonell, J. G., Knoblock, C. A., and Minton, S. (1989). PRODIGY: An integrated architecture for planning and learning. Technical report CMU-CS89-189, Computer Science Department, Carnegie-Mellon University.

Carbonell, J. R. and Collins, A. M. (1973). Natural semantics in artificial intelligence. In *IJCAI-73*, pp. 344–351.

Cardano, G. (1663). *Liber de ludo aleae*. Lyons.

Carnap, R. (1928). *Der logische Aufbau der Welt*. Weltkreis-verlag. Translated into English as (Carnap, 1967).

Carnap, R. (1948). On the application of inductive logic. *Philosophy and Phenomenological Research*, *8*, 133–148.

Carnap, R. (1950). *Logical Foundations of Probability*. University of Chicago Press.

Carroll, S. (2007). *The Making of the Fittest: DNA and the Ultimate Forensic Record of Evolution*. Norton.

Casati, R. and Varzi, A. (1999). *Parts and places: the structures of spatial representation*. MIT Press.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *AAAI-94*, pp. 1023– 1028.

Cassandras, C. G. and Lygeros, J. (2006). *Stochastic Hybrid Systems*. CRC Press.

Castro, R., Coates, M., Liang, G., Nowak, R., and Yu, B. (2004). Network tomography: Recent developments. *Statistical Science*, *19*(3), 499–517.

Cesa-Bianchi, N. and Lugosi, G. (2006). *Prediction, learning, and Games*. Cambridge University Press.

Cesta, A., Cortellessa, G., Denis, M., Donati, A., Fratini, S., Oddi, A., Policella, N., Rabenau, E., and Schulster, J. (2007). MEXAR2: AI solves mission planner problems. *IEEE Intelligent Systems*, *22*(4), 12–19.

Chakrabarti, P. P., Ghose, S., Acharya, A., and de Sarkar, S. C. (1989). Heuristic search in restricted memory. *AIJ*, *41*(2), 197–222.

Chandra, A. K. and Harel, D. (1980). Computable queries for relational data bases. *J. Computer and System Sciences*, *21*(2), 156–178.

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

Chapman, D. (1987). Planning for conjunctive goals. *AIJ*, *32*(3), 333–377.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press.

Charniak, E. (1996). Tree-bank grammars. In *AAAI-96*, pp. 1031–1036.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *AAAI97*, pp. 598–603.

Charniak, E. and Goldman, R. (1992). A Bayesian model of plan recognition. *AIJ*, *64*(1), 53–79.

Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence*. Addison-Wesley.

Charniak, E., Riesbeck, C., McDermott, D., and Meehan, J. (1987). *Artificial Intelligence Programming* (2nd edition). Lawrence Erlbaum Associates.

Charniak, E. (1991). Bayesian networks without tears. *AIMag*, *12*(4), 50–63.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL-05*.

Chater, N. and Oaksford, M. (Eds.). (2008). *The probabilistic mind: Prospects for Bayesian cognitive science*. Oxford University Press.

Chatfield, C. (1989). *The Analysis of Time Series: An Introduction* (4th edition). Chapman and Hall.

Cheeseman, P. (1985). In defense of probability. In *IJCAI-85*, pp. 1002–1009.

Cheeseman, P. (1988). An inquiry into computer understanding. *Computational Intelligence*, *4*(1), 58–66.

Cheeseman, P., Kanefsky, B., and Taylor, W. (1991). Where the really hard problems are. In *IJCAI-91*, pp. 331–337.

Cheeseman, P., Self, M., Kelly, J., and Stutz, J. (1988). Bayesian classification. In *AAAI-88*, Vol. 2, pp. 607–611.

Cheeseman, P. and Stutz, J. (1996). Bayesian classification (AutoClass): Theory and results. In Fayyad, U., Piatesky-Shapiro, G., Smyth, P., and Uthurusamy, R. (Eds.), *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press.

Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *ACL-96*, pp. 310–318.

Cheng, J. and Druzdzel, M. J. (2000). AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *JAIR*, *13*, 155–188.

Cheng, J., Greiner, R., Kelly, J., Bell, D. A., and Liu, W. (2002). Learning Bayesian networks from data: An information-theory based approach. *AIJ*, *137*, 43–90.

Chklovski, T. and Gil, Y. (2005). Improving the design of intelligent acquisition interfaces for collecting world knowledge from web contributors. In *Proc. Third International Conference on Knowledge Capture (K-CAP)*.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*(3), 113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton.

Choset, H. (1996). *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. Ph.D. thesis, California Institute of Technology.

Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S. (2004). *Principles of Robotic Motion: Theory, Algorithms, and Implementation*. MIT Press.

Chung, K. L. (1979). *Elementary Probability Theory with Stochastic Processes* (3rd edition). Springer-Verlag.

Church, A. (1936). A note on the Entscheidungsproblem. *JSL*, *1*, 40–41 and 101–102.

Church, A. (1956). *Introduction to Mathematical Logic*. Princeton University Press.

Church, K. and Patil, R. (1982). Coping with syntactic ambiguity or how to put the block in the box on the table. *Computational Linguistics*, *8*(3–4), 139–149.

Church, K. (2004). Speech and language processing: Can we use the past to predict the future. In *Proc. Conference on Text, Speech, and Dialogue*.

Church, K. and Gale, W. A. (1991). A comparison of the enhanced Good–Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, *5*, 19–54.

Churchland, P. M. and Churchland, P. S. (1982). Functionalism, qualia, and intentionality. In Biro, J. I. and Shahan, R. W. (Eds.), *Mind, Brain and Function: Essays in the Philosophy of Mind*, pp. 121–145. University of Oklahoma Press.

Churchland, P. S. (1986). *Neurophilosophy: Toward a Unified Science of the Mind–Brain*. MIT Press.

Ciancarini, P. and Wooldridge, M. (2001). *Agent-Oriented Software Engineering*. Springer-Verlag.

Cimatti, A., Roveri, M., and Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *AAAI-98*, pp. 875–881.

Clark, A. (1998). *Being There: Putting Brain, Body, and World Together Again*. MIT Press.

Clark, A. (2008). *Supersizing the Mind: Embodiment, Action, and Cognitive Extension*. Oxford University Press.

Clark, K. L. (1978). Negation as failure. In Gallaire, H. and Minker, J. (Eds.), *Logic and Data Bases*, pp. 293–322. Plenum.

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*, 261–283.

Clark, S. and Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *ACL-04*, pp. 104–111.

Clarke, A. C. (1968a). *2001: A Space Odyssey*. Signet.

Clarke, A. C. (1968b). The world of 2001. Vogue.

Clarke, E. and Grumberg, O. (1987). Research on automatic verification of finite-state concurrent systems. *Annual Review of Computer Science*, *2*, 269–290.

Clarke, M. R. B. (Ed.). (1977). *Advances in Computer Chess 1*. Edinburgh University Press.

Clearwater, S. H. (Ed.). (1996). *Market-Based Control*. World Scientific.

Clocksin, W. F. and Mellish, C. S. (2003). *Programming in Prolog* (5th edition). Springer-Verlag.

Clocksin, W. F. (2003). *Clause and Effect: Prolog Programming for the Working Programmer*. Springer.

Coarfa, C., Demopoulos, D., Aguirre, A., Subramanian, D., and Yardi, M. (2003). Random 3-SAT: The plot thickens. *Constraints*, *8*(3), 243–261.

Coates, A., Abbeel, P., and Ng, A. Y. (2009). Apprenticeship learning for helicopter control. *JACM*, *52*(7), 97–105.

Cobham, A. (1964). The intrinsic computational difficulty of functions. In *Proc. 1964 International Congress for Logic, Methodology, and Philosophy of Science*, pp. 24–30.

Cohen, P. R. (1995). *Empirical methods for artificial intelligence*. MIT Press.

Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *AIJ*, *42*(2–3), 213–261.

Cohen, P. R., Morgan, J., and Pollack, M. E. (1990). *Intentions in Communication*. MIT Press.

Cohen, W. W. and Page, C. D. (1995). Learnability in inductive logic programming: Methods and results. *New Generation Computing*, *13*(3–4), 369–409.

Cohn, A. G., Bennett, B., Gooday, J. M., and Gotts, N. (1997). RCC: A calculus for region based qualitative spatial reasoning. *GeoInformatica*, *1*, 275–316.

Collin, Z., Dechter, R., and Katz, S. (1999). Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*, *1999*(115).

Collins, F. S., Morgan, M., and Patrinos, A. (2003). The human genome project: Lessons from large-scale biology. *Science*, *300*(5617), 286–290.

Collins, M. (1999). *Head-driven Statistical Models for Natural Language Processing*. Ph.D. thesis, University of Pennsylvania.

Collins, M. and Duffy, K. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL-02*.

Colmerauer, A. and Roussel, P. (1993). The birth of Prolog. *SIGPLAN Notices*, *28*(3), 37–52.

Colmerauer, A. (1975). Les grammaires de metamorphose. Tech. rep., Groupe d'Intelligence Artificielle, Universit´e de Marseille-Luminy.

Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P. (1973). Un syst´eme de communication homme–machine en Franc¸ais. Rapport, Groupe d'Intelligence Artificielle, Universit´e d'Aix-Marseille II.

Condon, J. H. and Thompson, K. (1982). Belle chess hardware. In Clarke, M. R. B. (Ed.), *Advances in Computer Chess 3*, pp. 45–54. Pergamon.

Congdon, C. B., Huber, M., Kortenkamp, D., Bid-lack, C., Cohen, C., Huffman, S., Koss, F., Raschke, U., and Weymouth, T. (1992). CARMEL versus Flakey: A comparison of two robots. Tech. rep. Papers from the AAAI Robot Competition, RC-92-01, American Association for Artificial Intelligence.

Conlisk, J. (1989). Three variants on the Allais example. *American Economic Review*, *79*(3), 392–407.

Connell, J. (1989). *A Colony Architecture for an Artificial Creature*. Ph.D. thesis, Artificial Intelligence Laboratory, MIT. Also available as AI Technical Report 1151.

Consortium, T. G. O. (2008). The gene ontology project in 2008. *Nucleic Acids Research*, *36*.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *STOC-71*, pp. 151–158.

Cook, S. A. and Mitchell, D. (1997). Finding hard instances of the satisfiability problem: A survey. In Du, D., Gu, J., and Pardalos, P. (Eds.), *Satisfiability problems: Theory and applications*. American Mathematical Society.

Cooper, G. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *AIJ*, *42*, 393–405.

Cooper, G. and Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

Copeland, J. (1993). *Artificial Intelligence: A Philosophical Introduction*. Blackwell.

Copernicus (1543). *De Revolutionibus Orbium Coelestium*. Apud Ioh. Petreium, Nuremberg.

Cormen, T. H., Leiserson, C. E., and Rivest, R. (1990). *Introduction to Algorithms*. MIT Press.

Cortes, C. and Vapnik, V. N. (1995). Support vector networks. *Machine Learning*, *20*, 273–297.

Cournot, A. (Ed.). (1838). *Recherches sur les principes math´ematiques de la th´eorie des richesses*. L. Hachette, Paris.

Cover, T. and Thomas, J. (2006). *Elements of Information Theory* (2nd edition). Wiley.

Cowan, J. D. and Sharp, D. H. (1988a). Neural nets. *Quarterly Reviews of Biophysics*, *21*, 365–427.

Cowan, J. D. and Sharp, D. H. (1988b). Neural nets and artificial intelligence. *Daedalus*, *117*, 85–121.

Cowell, R., Dawid, A. P., Lauritzen, S., and Spiegelhalter, D. J. (2002). *Probabilistic Networks and Expert Systems*. Springer.

Cox, I. (1993). A review of statistical data association techniques for motion correspondence. *IJCV*, *10*, 53–66.

Cox, I. and Hingorani, S. L. (1994). An efficient implementation and evaluation of Reid's multiple hypothesis tracking algorithm for visual tracking. In *ICPR-94*, Vol. 1, pp. 437–442.

Cox, I. and Wilfong, G. T. (Eds.). (1990). *Autonomous Robot Vehicles*. Springer Verlag.

Cox, R. T. (1946). Probability, frequency, and reasonable expectation. *American Journal of Physics*,

14(1), 1–13.

Craig, J. (1989). *Introduction to Robotics: Mechanics and Control (2nd edition)*. Addison-Wesley Publishing, Inc.

Craik, K. J. (1943). *The Nature of Explanation*. Cambridge University Press.

Craswell, N., Zaragoza, H., and Robertson, S. E. (2005). Microsoft cambridge at trec-14: Enterprise track. In *Proc. Fourteenth Text REtrieval Conference*.

Crauser, A., Mehlhorn, K., Meyer, U., and Sanders, P. (1998). A parallelization of Dijkstra's shortest path algorithm. In *Proc. 23rd International Symposium on Mathematical Foundations of Computer Science,*, pp. 722–731.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T. M., Nigam, K., and Slattery, S. (2000). Learning to construct knowledge bases from the World Wide Web. *AIJ*, *118*(1/2), 69–113.

Crawford, J. M. and Auton, L. D. (1993). Experimental results on the crossover point in satisfiability problems. In *AAAI-93*, pp. 21–27.

Cristianini, N. and Hahn, M. (2007). *Introduction to Computational Genomics: A Case Studies Approach*. Cambridge University Press.

Cristianini, N. and Sch¨olkopf, B. (2002). Support vector machines and kernel methods: The new generation of learning machines. *AIMag*, *23*(3), 31–41.

Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press.

Crockett, L. (1994). *The Turing Test and the Frame Problem: AI's Mistaken Understanding of Intelligence*. Ablex.

Croft, B., Metzler, D., and Stroham, T. (2009). *Search Engines: Information retrieval in Practice*. Addison Wesley.

Cross, S. E. and Walker, E. (1994). DART: Applying knowledge based planning and scheduling to crisis action planning. In Zweben, M. and Fox, M. S. (Eds.), *Intelligent Scheduling*, pp. 711–729. Morgan Kaufmann.

Cruse, D. A. (1986). *Lexical Semantics*. Cambridge University Press.

Culberson, J. and Schaeffer, J. (1996). Searching with pattern databases. In *Advances in Artificial Intelligence (Lecture Notes in Artificial Intelligence 1081)*, pp. 402–416. Springer-Verlag.

Culberson, J. and Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, *14*(4), 318–334.

Cullingford, R. E. (1981). Integrating knowledge sources for computer "understanding" tasks. *IEEE Transactions on Systems, Man and Cybernetics (SMC)*, *11*.

Cummins, D. and Allen, C. (1998). *The Evolution of Mind*. Oxford University Press.

Cushing, W., Kambhampati, S., Mausam, and Weld, D. S. (2007). When is temporal planning *really* temporal? In *IJCAI-07*.

Cybenko, G. (1988). Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Controls, Signals, and Systems*, *2*, 303–314.

Daganzo, C. (1979). *Multinomial probit: The theory and its application to demand forecasting*. Academic Press.

Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *AIJ*, *60*(1), 141–153.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *CVPR*, pp. 886–893.

Dantzig, G. B. (1949). Programming of interdependent activities: II. Mathematical model. *Econometrica*, *17*, 200–211.

Darwiche, A. (2001). Recursive conditioning. *AIJ*, *126*, 5–41.

Darwiche, A. and Ginsberg, M. L. (1992). A symbolic generalization of probability theory. In *AAAI92*, pp. 622–627.

Darwiche, A. (2009). *Modeling and reasoning with Bayesian networks*. Cambridge University Press.

Darwin, C. (1859). *On The Origin of Species by Means of Natural Selection*. J. Murray, London.

Darwin, C. (1871). *Descent of Man*. J. Murray.

Dasgupta, P., Chakrabarti, P. P., and de Sarkar, S. C. (1994). Agent searching in a tree and the optimality of iterative deepening. *AIJ*, *71*, 195–208.

Davidson, D. (1980). *Essays on Actions and Events*. Oxford University Press.

Davies, T. R. (1985). Analogy. Informal note INCSLI-85-4, Center for the Study of Language and Information (CSLI).

Davies, T. R. and Russell, S. J. (1987). A logical approach to reasoning by analogy. In *IJCAI-87*, Vol. 1, pp. 264–270.

Davis, E. (1986). *Representing and Acquiring Geographic Knowledge*. Pitman and Morgan Kaufmann.

Davis, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann.

Davis, E. (2005). Knowledge and communication: A first-order theory. *AIJ*, *166*, 81–140.

Davis, E. (2006). The expressivity of quantifying over regions. *J. Logic and Computation*, *16*, 891–916.

Davis, E. (2007). Physical reasoning. In van Harmelan, F., Lifschitz, V., and Porter, B. (Eds.), *The Handbook of Knowledge Representation*, pp. 597–620. Elsevier.

Davis, E. (2008). Pouring liquids: A study in commonsense physical reasoning. *AIJ*, *172*(1540–

1578).

Davis, E. and Morgenstern, L. (2004). Introduction: Progress in formal commonsense reasoning. *AIJ*, *153*, 1–12.

Davis, E. and Morgenstern, L. (2005). A first-order theory of communication and multi-agent plans. *J. Logic and Computation*, *15*(5), 701–749.

Davis, K. H., Biddulph, R., and Balashek, S. (1952). Automatic recognition of spoken digits. *J. Acoustical Society of America*, *24*(6), 637–642.

Davis, M. (1957). A computer program for Presburger's algorithm. In *Proving Theorems (as Done by Man, Logician, or Machine)*, pp. 215–233. Proc. Summer Institute for Symbolic Logic. Second edition; publication date is 1960.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *CACM*, *5*, 394–397.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *JACM*, *7*(3), 201–215.

Davis, R. and Lenat, D. B. (1982). *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning*, *8*(3–4), 341–362.

Dayan, P. and Abbott, L. F. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press.

Dayan, P. and Niv, Y. (2008). Reinforcement learning and the brain: The good, the bad and the ugly. *Current Opinion in Neurobiology*, *18*(2), 185–196.

de Dombal, F. T., Leaper, D. J., Horrocks, J. C., and Staniland, J. R. (1974). Human and computer-aided diagnosis of abdominal pain: Further report with emphasis on performance of clinicians. *British Medical Journal*, *1*, 376–380.

de Dombal, F. T., Staniland, J. R., and Clamp, S. E. (1981). Geographical variation in disease presentation. *Medical Decision Making*, *1*, 59–69.

de Finetti, B. (1937). Le pr´evision: ses lois logiques, ses sources subjectives. *Ann. Inst. Poincaré*, *7*, 1–68.

de Finetti, B. (1993). On the subjective meaning of probability. In Monari, P. and Cocchi, D. (Eds.), *Probabilita e Induzione*, pp. 291–321. Clueb.

de Freitas, J. F. G., Niranjan, M., and Gee, A. H. (2000). Sequential Monte Carlo methods to train neural network models. *Neural Computation*, *12*(4), 933–953.

de Kleer, J. (1975). Qualitative and quantitative knowledge in classical mechanics. Tech. rep. AITR-352, MIT Artificial Intelligence Laboratory.

de Kleer, J. (1989). A comparison of ATMS and CSP techniques. In *IJCAI-89*, Vol. 1, pp. 290–296.

de Kleer, J. and Brown, J. S. (1985). A qualitative physics based on confluences. In Hobbs, J. R. and

Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 4, pp. 109–183. Ablex.

de Marcken, C. (1996). *Unsupervised Language Acquisition*. Ph.D. thesis, MIT.

De Morgan, A. (1864). On the syllogism, No. IV, and on the logic of relations. *Transaction of the Cambridge Philosophical Society*, *X*, 331–358.

De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press.

de Salvo Braz, R., Amir, E., and Roth, D. (2007). Lifted first-order probabilistic inference. In Getoor, L. and Taskar, B. (Eds.), *Introduction to Statistical Relational Learning*. MIT Press.

Deacon, T. W. (1997). *The symbolic species: The coevolution of language and the brain*. W. W. Norton.

Deale, M., Yvanovich, M., Schnitzius, D., Kautz, D., Carpenter, M., Zweben, M., Davis, G., and Daun, B. (1994). The space shuttle ground processing scheduling system. In Zweben, M. and Fox, M. (Eds.), *Intelligent Scheduling*, pp. 423–449. Morgan Kaufmann.

Dean, T., Basye, K., Chekaluk, R., and Hyun, S. (1990). Coping with uncertainty in a control system for navigation and exploration. In *AAAI-90*, Vol. 2, pp. 1010–1015.

Dean, T. and Boddy, M. (1988). An analysis of time-dependent planning. In *AAAI-88*, pp. 49–54.

Dean, T., Firby, R. J., and Miller, D. (1990). Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, *6*(1), 381–398.

Dean, T., Kaelbling, L. P., Kirman, J., and Nicholson, A. (1993). Planning with deadlines in stochastic domains. In *AAAI-93*, pp. 574–579.

Dean, T. and Kanazawa, K. (1989a). A model for projection and action. In *IJCAI-89*, pp. 985–990.

Dean, T. and Kanazawa, K. (1989b). A model for reasoning about persistence and causation. *Computational Intelligence*, *5*(3), 142–150.

Dean, T., Kanazawa, K., and Shewchuk, J. (1990). Prediction, observation and estimation in planning and control. In *5th IEEE International Symposium on Intelligent Control*, Vol. 2, pp. 645–650.

Dean, T. and Wellman, M. P. (1991). *Planning and Control*. Morgan Kaufmann.

Dearden, R., Friedman, N., and Andre, D. (1999). Model-based Bayesian exploration. In *UAI-99*.

Dearden, R., Friedman, N., and Russell, S. J. (1998). Bayesian q-learning. In *AAAI-98*.

Debevec, P., Taylor, C., and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proc. 23rd Annual Conference on Computer Graphics (SIGGRAPH)*, pp. 11–20.

Debreu, G. (1960). Topological methods in cardinal utility theory. In Arrow, K. J., Karlin, S., and Sup-pes, P. (Eds.), *Mathematical Methods in the Social Sciences, 1959*. Stanford University Press.

Dechter, R. (1990a). Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *AIJ*, *41*, 273–312.

Dechter, R. (1990b). On the expressiveness of networks with hidden variables. In *AAAI-90*, pp. 379–385.

Dechter, R. (1992). Constraint networks. In Shapiro, S. (Ed.), *Encyclopedia of Artificial Intelligence* (2nd edition)., pp. 276–285. Wiley and Sons.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *AIJ*, *113*, 41–85.

Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *JACM*, *32*(3), 505–536.

Dechter, R. and Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *AIJ*, *34*(1), 1–38.

Dechter, R. and Pearl, J. (1989). Tree clustering for constraint networks. *AIJ*, *38*(3), 353–366.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Dechter, R. and Frost, D. (2002). Backjump-based backtracking for constraint satisfaction problems. *AIJ*, *136*(2), 147–188.

Dechter, R. and Mateescu, R. (2007). AND/OR search spaces for graphical models. *AIJ*, *171*(2–3), 73–106.

DeCoste, D. and Sch¨olkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, *46*(1), 161–190.

Dedekind, R. (1888). *Was sind und was sollen die Zahlen*. Braunschweig, Germany.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *J. American Society for Information Science*, *41*(6), 391–407.

DeGroot, M. H. (1970). *Optimal Statistical Decisions*. McGraw-Hill.

DeGroot, M. H. and Schervish, M. J. (2001). *Probability and Statistics* (3rd edition). Addison Wesley.

DeJong, G. (1981). Generalizations based on explanations. In *IJCAI-81*, pp. 67–69.

DeJong, G. (1982). An overview of the FRUMP system. In Lehnert, W. and Ringle, M. (Eds.), *Strategies for Natural Language Processing*, pp. 149–176. Lawrence Erlbaum.

DeJong, G. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, *1*, 145–176.

Del Moral, P., Doucet, A., and Jasra, A. (2006). Sequential Monte Carlo samplers. *J. Royal Statistical Society, Series B*, *68*(3), 411–436.

Del Moral, P. (2004). *Feynman–Kac Formulae, Genealogical and Interacting Particle Systems with Applications*. Springer-Verlag.

Delgrande, J. and Schaub, T. (2003). On the relation between Reiter's default logic and its (major) variants. In *Seventh European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pp. 452–463.

Dempster, A. P. (1968). A generalization of Bayesian inference. *J. Royal Statistical Society*, *30 (Series B)*, 205–247.

Dempster, A. P., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society*, *39 (Series B)*, 1–38.

Deng, X. and Papadimitriou, C. H. (1990). Exploring an unknown graph. In *FOCS-90*, pp. 355–361.

Denis, F. (2001). Learning regular languages from simple positive examples. *Machine Learning*, *44*(1/2), 37–66.

Dennett, D. C. (1984). Cognitive wheels: the frame problem of AI. In Hookway, C. (Ed.), *Minds, Machines, and Evolution: Philosophical Studies*, pp. 129–151. Cambridge University Press.

Dennett, D. C. (1991). *Consciousness Explained*. Penguin Press.

Denney, E., Fischer, B., and Schumann, J. (2006). An empirical evaluation of automated theorem provers in software certification. *Int. J. AI Tools*, *15*(1), 81–107.

Descartes, R. (1637). Discourse on method. In Cottingham, J., Stoothoff, R., and Murdoch, D. (Eds.), *The Philosophical Writings of Descartes*, Vol. I. Cambridge University Press, Cambridge, UK.

Descartes, R. (1641). Meditations on first philosophy. In Cottingham, J., Stoothoff, R., and Murdoch, D. (Eds.), *The Philosophical Writings of Descartes*, Vol. II. Cambridge University Press, Cambridge, UK.

Descotte, Y. and Latombe, J.-C. (1985). Making compromises among antagonist constraints in a planner. *AIJ*, *27*, 183–217.

Detwarasiti, A. and Shachter, R. D. (2005). Influence diagrams for team decision analysis. *Decision Analysis*, *2*(4), 207–228.

Devroye, L. (1987). *A course in density estimation*. Birkhauser.

Dickmanns, E. D. and Zapp, A. (1987). Autonomous high speed road vehicle guidance by computer vision. In *Automatic Control—World Congress, 1987: Selected Papers from the 10th Triennial World Congress of the International Federation of Automatic Control*, pp. 221–226.

Dietterich, T. (1990). Machine learning. *Annual Review of Computer Science*, *4*, 255–306.

Dietterich, T. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *JAIR*, *13*, 227–303.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271.

Dijkstra, E. W. (1984). The threats to computing science. In *ACM South Central Regional Conference*.

Dillenburg, J. F. and Nelson, P. C. (1994). Perimeter search. *AIJ*, *65*(1), 165–178.

Dinh, H., Russell, A., and Su, Y. (2007). On the value of good advice: The complexity of A* with accurate heuristics. In *AAAI-07*.

Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H., and Csorba, M. (2001). A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, *17*(3), 229–241.

Do, M. B. and Kambhampati, S. (2001). Sapa: A domain-independent heuristic metric temporal planner. In *ECP-01*.

Do, M. B. and Kambhampati, S. (2003). Planning as constraint satisfaction: solving the planning graph by compiling it into CSP. *AIJ*, *132*(2), 151–182.

Doctorow, C. (2001). Metacrap: Putting the torch to seven straw-men of the meta-utopia. www.well.com/~doctorow/metacrap.htm.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero– one loss. *Machine Learning*, *29*, 103–30.

Domingos, P. and Richardson, M. (2004). Markov logic: A unifying framework for statistical relational learning. In *Proc. ICML-04 Workshop on Statistical Relational Learning*.

Donninger, C. and Lorenz, U. (2004). The chess monster hydra. In *Proc. 14th International Conference on Field-Programmable Logic and Applications*, pp. 927–932.

Doorenbos, R. (1994). Combining left and right unlinking for matching a large number of learned rules. In *AAAI-94*.

Doran, J. and Michie, D. (1966). Experiments with the graph traverser program. *Proc. Royal Society of London*, *294, Series A*, 235–259.

Dorf, R. C. and Bishop, R. H. (2004). *Modern Control Systems* (10th edition). Prentice-Hall.

Doucet, A. (1997). *Monte Carlo methods for Bayesian estimation of hidden Markov models: Application to radiation signals*. Ph.D. thesis, Universit´e de Paris-Sud.

Doucet, A., de Freitas, N., and Gordon, N. (2001). *Sequential Monte Carlo Methods in Practice*. Springer-Verlag.

Doucet, A., de Freitas, N., Murphy, K., and Russell, S. J. (2000). Rao-blackwellised particle filtering for dynamic bayesian networks. In *UAI-00*.

Dowling, W. F. and Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulas. *J. Logic Programming*, *1*, 267–284.

Dowty, D., Wall, R., and Peters, S. (1991). *Introduction to Montague Semantics*. D. Reidel.

Doyle, J. (1979). A truth maintenance system. *AIJ*, *12*(3), 231–272.

Doyle, J. (1983). What is rational psychology? Toward a modern mental philosophy. *AIMag*, *4*(3), 50–53.

Doyle, J. and Patil, R. (1991). Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *AIJ*, *48*(3), 261–297.

Drabble, B. (1990). Mission scheduling for spacecraft: Diaries of T-SCHED. In *Expert Planning Systems*, pp. 76–81. Institute of Electrical Engineers.

Dredze, M., Crammer, K., and Pereira, F. (2008). Confidence-weighted linear classification. In *ICML08*, pp. 264–271.

Dreyfus, H. L. (1972). *What Computers Can't Do: A Critique of Artificial Reason*. Harper and Row.

Dreyfus, H. L. (1992). *What Computers Still Can't Do: A Critique of Artificial Reason*. MIT Press.

Dreyfus, H. L. and Dreyfus, S. E. (1986). *Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. Blackwell.

Dreyfus, S. E. (1969). An appraisal of some shortest-paths algorithms. *Operations Research*, *17*, 395–412.

Dubois, D. and Prade, H. (1994). A survey of belief revision and updating rules in various uncertainty models. *Int. J. Intelligent Systems*, *9*(1), 61–100.

Duda, R. O., Gaschnig, J., and Hart, P. E. (1979). Model design in the Prospector consultant system for mineral exploration. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age*, pp. 153–167. Edinburgh University Press.

Duda, R. O. and Hart, P. E. (1973). *Pattern classification and scene analysis*. Wiley.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification* (2nd edition). Wiley.

Dudek, G. and Jenkin, M. (2000). *Computational Principles of Mobile Robotics*. Cambridge University Press.

Duffy, D. (1991). *Principles of Automated Theorem Proving*. John Wiley & Sons.

Dunn, H. L. (1946). Record linkage". *Am. J. Public Health*, *36*(12), 1412–1416.

Durfee, E. H. and Lesser, V. R. (1989). Negotiating task decomposition and allocation using partial global planning. In Huhns, M. and Gasser, L. (Eds.), *Distributed AI*, Vol. 2. Morgan Kaufmann.

Durme, B. V. and Pasca, M. (2008). Finding cars, goddesses and enzymes: Parametrizable acquisition of labeled instances for open-domain information extraction. In *AAAI-08*, pp. 1243–1248.

Dyer, M. (1983). *In-Depth Understanding*. MIT Press.

Dyson, G. (1998). *Darwin among the machines : the evolution of global intelligence*. Perseus Books.

Duzeroski, S., Muggleton, S. H., and Russell, S. J. (1992). PAC-learnability of determinate logic programs. In *COLT-92*, pp. 128–135.

Earley, J. (1970). An efficient context-free parsing algorithm. *CACM*, *13*(2), 94–102.

Edelkamp, S. (2009). Scaling search with symbolic pattern databases. In *Model Checking and Artificial Intelligence (MOCHART)*, pp. 49–65.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics*, *17*, 449–467.

Edwards, P. (Ed.). (1967). *The Encyclopedia of Philosophy*. Macmillan.

Een, N. and S¨orensson, N. (2003). An extensible SAT-solver. In Giunchiglia, E. and Tacchella, A. (Eds.), *Theory and Applications of Satisfiability Testing: 6th International Conference (SAT 2003)*. Springer-Verlag.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1998). The KR system dlv: Progress report, comparisons and benchmarks. In *KR-98*, pp. 406–417.

Elio, R. (Ed.). (2002). *Common Sense, Reasoning, and Rationality*. Oxford University Press.

Elkan, C. (1993). The paradoxical success of fuzzy logic. In *AAAI-93*, pp. 698–703.

Elkan, C. (1997). Boosting and naive Bayesian learning. Tech. rep., Department of Computer Science and Engineering, University of California, San Diego.

Ellsberg, D. (1962). *Risk, Ambiguity, and Decision*. Ph.D. thesis, Harvard University.

Elman, J., Bates, E., Johnson, M., Karmiloff-Smith, A., Parisi, D., and Plunkett, K. (1997). *Rethinking Innateness*. MIT Press.

Empson, W. (1953). *Seven Types of Ambiguity*. New Directions.

Enderton, H. B. (1972). *A Mathematical Introduction to Logic*. Academic Press.

Epstein, R., Roberts, G., and Beber, G. (Eds.). (2008). *Parsing the Turing Test*. Springer.

Erdmann, M. A. and Mason, M. (1988). An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, *4*(4), 369–379.

Ernst, H. A. (1961). *MH-1, a Computer-Operated Mechanical Hand*. Ph.D. thesis, Massachusetts Institute of Technology.

Ernst, M., Millstein, T., and Weld, D. S. (1997). Automatic SAT-compilation of planning problems. In *IJCAI-97*, pp. 1169–1176.

Erol, K., Hendler, J., and Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *AAAI-94*, pp. 1123–1128.

Erol, K., Hendler, J., and Nau, D. S. (1996). Complexity results for HTN planning. *AIJ*, *18*(1), 69–93.

Etzioni, A. (2004). *From Empire to Community: A New Approach to International Relation*. Palgrave Macmillan.

Etzioni, O. (1989). Tractable decision-analytic control. In *Proc. First International Conference on Knowledge Representation and Reasoning*, pp. 114–125.

Etzioni, O., Banko, M., Soderland, S., and Weld, D. S. (2008). Open information extraction from the web. *CACM*, *51*(12).

Etzioni, O., Hanks, S., Weld, D. S., Draper, D., Lesh, N., and Williamson, M. (1992). An approach to planning with incomplete information. In *KR-92*.

Etzioni, O. and Weld, D. S. (1994). A softbot-based interface to the Internet. *CACM*, *37*(7), 72–76.

Etzioni, O., Banko, M., and Cafarella, M. J. (2006). Machine reading. In *AAAI-06*.

Etzioni, O., Cafarella, M. J., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S., and Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *AIJ*, *165*(1), 91–134.

Evans, T. G. (1968). A program for the solution of a class of geometric-analogy intelligence-test questions. In Minsky, M. L. (Ed.), *Semantic Information Processing*, pp. 271–353. MIT Press.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning about Knowledge*. MIT Press.

Fahlman, S. E. (1974). A planning system for robot construction tasks. *AIJ*, *5*(1), 1–49.

Faugeras, O. (1993). *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press.

Faugeras, O., Luong, Q.-T., and Papadopoulo, T. (2001). *The Geometry of Multiple Images*. MIT Press.

Fearing, R. S. and Hollerbach, J. M. (1985). Basic solid mechanics for tactile sensing. *Int. J. Robotics Research*, *4*(3), 40–54.

Featherstone, R. (1987). *Robot Dynamics Algorithms*. Kluwer Academic Publishers.

Feigenbaum, E. A. (1961). The simulation of verbal learning behavior. *Proc. Western Joint Computer Conference*, *19*, 121–131.

Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J. (1971). On generality and problem solving: A case study using the DENDRAL program. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 6*, pp. 165–190. Edinburgh University Press.

Feldman, J. and Sproull, R. F. (1977). Decision theory and artificial intelligence II: The hungry monkey. Technical report, Computer Science Department, University of Rochester.

Feldman, J. and Yakimovsky, Y. (1974). Decision theory and artificial intelligence I: Semantics-based region analyzer. *AIJ*, *5*(4), 349–371.

Fellbaum, C. (2001). *Wordnet: An Electronic Lexical Database*. MIT Press.

Fellegi, I. and Sunter, A. (1969). A theory for record linkage". *JASA*, *64*, 1183–1210.

Felner, A., Korf, R. E., and Hanan, S. (2004). Additive pattern database heuristics. *JAIR*, *22*, 279–318.

Felner, A., Korf, R. E., Meshulam, R., and Holte, R. (2007). Compressed pattern databases. *JAIR*, *30*, 213–247.

Felzenszwalb, P. and Huttenlocher, D. (2000). Efficient matching of pictorial structures. In *CVPR*.

Felzenszwalb, P. and McAllester, D. A. (2007). The generalized A* architecture. *JAIR*.

Ferguson, T. (1992). Mate with knight and bishop in kriegspiel. *Theoretical Computer Science*, *96*(2), 389–403.

Ferguson, T. (1995). Mate with the two bishops in kriegspiel. www.math.ucla.edu/~tom/papers.

Ferguson, T. (1973). Bayesian analysis of some nonparametric problems. *Annals of Statistics*, *1*(2),

209–230.

Ferraris, P. and Giunchiglia, E. (2000). Planning as satisability in nondeterministic domains. In *AAAI00*, pp. 748–753.

Ferriss, T. (2007). *The 4-Hour Workweek*. Crown.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *AIJ*, *3*(4), 251–288.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *AIJ*, *2*(3–4), 189–208.

Fikes, R. E. and Nilsson, N. J. (1993). STRIPS, a retrospective. *AIJ*, *59*(1–2), 227–232.

Fine, S., Singer, Y., and Tishby, N. (1998). The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, *32*(41–62).

Finney, D. J. (1947). *Probit analysis: A statistical treatment of the sigmoid response curve*. Cambridge University Press.

Firth, J. (1957). *Papers in Linguistics*. Oxford University Press.

Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London*, *Series A 222*, 309–368.

Fix, E. and Hodges, J. L. (1951). Discriminatory analysis—Nonparametric discrimination: Consistency properties. Tech. rep. 21-49-004, USAF School of Aviation Medicine.

Floreano, D., Zufferey, J. C., Srinivasan, M. V., and Ellington, C. (2009). *Flying Insects and Robots*. Springer.

Fogel, D. B. (2000). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. Wiley.

Foo, N. (2001). Why engineering models do not have a frame problem. In *Discrete event modeling and simulation technologies: a tapestry of systems and AI-based theories and methodologies*. Springer.

Forbes, J. (2002). *Learning Optimal Control for Autonomous Vehicles*. Ph.D. thesis, University of California.

Forbus, K. D. (1985). Qualitative process theory. In Bobrow, D. (Ed.), *Qualitative Reasoning About Physical Systems*, pp. 85–186. MIT Press.

Forbus, K. D. and de Kleer, J. (1993). *Building Problem Solvers*. MIT Press.

Ford, K. M. and Hayes, P. J. (1995). Turing Test considered harmful. In *IJCAI-95*, pp. 972–977.

Forestier, J.-P. and Varaiya, P. (1978). Multilayer control of large Markov chains. *IEEE Transactions on Automatic Control*, *23*(2), 298–304.

Forgy, C. (1981). OPS5 user's manual. Technical report CMU-CS-81-135, Computer Science Department, Carnegie-Mellon University.

Forgy, C. (1982). A fast algorithm for the many patterns/many objects match problem. *AIJ*, *19*(1), 17–37.

Forsyth, D. and Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall.

Fourier, J. (1827). Analyse des travaux de l'Acad´emie Royale des Sciences, pendant l'ann´ee 1824; partie math´ematique. *Histoire de l'Acad´emie Royale des Sciences de France*, *7*, xlvii–lv.

Fox, C. and Tversky, A. (1995). Ambiguity aversion and comparative ignorance. *Quarterly Journal of Economics*, *110*(3), 585–603.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. In *AAAI-99*.

Fox, M. S. (1990). Constraint-guided scheduling: A short history of research at CMU. *Computers in Industry*, *14*(1–3), 79–88.

Fox, M. S., Allen, B., and Strohm, G. (1982). Job shop scheduling: An investigation in constraint-directed reasoning. In *AAAI-82*, pp. 155–158.

Fox, M. S. and Long, D. (1998). The automatic inference of state invariants in TIM. *JAIR*, *9*, 367–421.

Franco, J. and Paull, M. (1983). Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, *5*, 77–87.

Frank, I., Basin, D. A., and Matsubara, H. (1998). Finding optimal strategies for imperfect information games. In *AAAI-98*, pp. 500–507.

Frank, R. H. and Cook, P. J. (1996). *The WinnerTake-All Society*. Penguin.

Franz, A. (1996). *Automatic Ambiguity resolution in Natural Language Processing: An Empirical Approach*. Springer.

Franz, A. and Brants, T. (2006). All our n-gram are belong to you. Blog posting.

Frege, G. (1879). *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, Berlin. English translation appears in van Heijenoort (1967).

Freitag, D. and McCallum, A. (2000). Information extraction with hmm structures learned by stochastic optimization. In *AAAI-00*.

Freuder, E. C. (1978). Synthesizing constraint expressions. *CACM*, *21*(11), 958–966.

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *JACM*, *29*(1), 24–32.

Freuder, E. C. (1985). A sufficient condition for backtrack-bounded search. *JACM*, *32*(4), 755–761.

Freuder, E. C. and Mackworth, A. K. (Eds.). (1994). *Constraint-based reasoning*. MIT Press.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *ICML-96*.

Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, *37*(3), 277–296.

Friedberg, R. M. (1958). A learning machine: Part I. *IBM Journal of Research and Development*, *2*, 2–13.

Friedberg, R. M., Dunham, B., and North, T. (1959). A learning machine: Part II. *IBM Journal of Research and Development*, *3*(3), 282–287.

Friedgut, E. (1999). Necessary and sufficient conditions for sharp thresholds of graph properties, and the k-SAT problem. *J. American Mathematical Society*, *12*, 1017–1054.

Friedman, G. J. (1959). Digital simulation of an evolutionary process. *General Systems Yearbook*, *4*, 171–184.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, *28*(2), 337–374.

Friedman, N. (1998). The Bayesian structural EM algorithm. In *UAI-98*.

Friedman, N. and Goldszmidt, M. (1996). Learning Bayesian networks with local structure. In *UAI-96*, pp. 252–262.

Friedman, N. and Koller, D. (2003). Being Bayesian about Bayesian network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, *50*, 95–125.

Friedman, N., Murphy, K., and Russell, S. J. (1998). Learning the structure of dynamic probabilistic networks. In *UAI-98*.

Friedman, N. (2004). Inferring cellular networks using probabilistic graphical models. *Science*, *303*(5659), 799–805.

Fruhwirth, T. and Abdennadher, S. (2003). *Essentials of constraint programming*. Cambridge University Press.

Fuchs, J. J., Gasquet, A., Olalainty, B., and Currie, K. W. (1990). PlanERS-1: An expert planning system for generating spacecraft mission plans. In *First International Conference on Expert Planning Systems*, pp. 70–75. Institute of Electrical Engineers.

Fudenberg, D. and Tirole, J. (1991). *Game theory*. MIT Press.

Fukunaga, A. S., Rabideau, G., Chien, S., and Yan, D. (1997). ASPEN: A framework for automated planning and scheduling of spacecraft control and operations. In *Proc. International Symposium on AI, Robotics and Automation in Space*, pp. 181–187.

Fung, R. and Chang, K. C. (1989). Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *UAI-98*, pp. 209–220.

Gaddum, J. H. (1933). Reports on biological standard III: Methods of biological assay depending on a quantal response. Special report series of the medical research council 183, Medical Research Council.

Gaifman, H. (1964). Concerning measures in first order calculi. *Israel Journal of Mathematics*, *2*,

1–18.

Gallaire, H. and Minker, J. (Eds.). (1978). *Logic and Databases*. Plenum.

Gallier, J. H. (1986). *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper and Row.

Gamba, A., Gamberini, L., Palmieri, G., and Sanna, R. (1961). Further experiments with PAPA. *Nuovo Cimento Supplemento*, *20*(2), 221–231.

Garding, J. (1992). Shape from texture for smooth curved surfaces in perspective projection. *J. Mathematical Imaging and Vision*, *2*(4), 327–350.

Gardner, M. (1968). *Logic Machines, Diagrams and Boolean Algebra*. Dover.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability*. W. H. Freeman.

Gaschnig, J. (1977). A general backtrack algorithm that eliminates most redundant tests. In *IJCAI-77*, p. 457.

Gaschnig, J. (1979). Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124, Computer Science Department, Carnegie-Mellon University.

Gasser, R. (1995). *Efficiently harnessing computational resources for exhaustive search*. Ph.D. thesis, ETH Zürich.

Gasser, R. (1998). Solving nine men's morris. In Nowakowski, R. (Ed.), *Games of No Chance*. Cambridge University Press.

Gat, E. (1998). Three-layered architectures. In Kortenkamp, D., Bonasso, R. P., and Murphy, R. (Eds.), *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, pp. 195–210. MIT Press.

Gauss, C. F. (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. Sumtibus F. Perthes et I. H. Besser, Hamburg.

Gauss, C. F. (1829). Beiträge zur theorie der algebraischen gleichungen. Collected in *Werke, Vol. 3*, pages 71–102. K. Gesellschaft Wissenschaft, Göttingen, Germany, 1876.

Gawande, A. (2002). *Complications: A Surgeon's Notes on an Imperfect Science*. Metropolitan Books.

Geiger, D., Verma, T., and Pearl, J. (1990). Identifying independence in Bayesian networks. *Networks*, *20*(5), 507–534.

Geisel, T. (1955). *On Beyond Zebra*. Random House.

Gelb, A. (1974). *Applied Optimal Estimation*. MIT Press.

Gelernter, H. (1959). Realization of a geometry-theorem proving machine. In *Proc. an International Conference on Information Processing*, pp. 273–282. UNESCO House.

Gelfond, M. and Lifschitz, V. (1988). Compiling circumscriptive theories into logic programs. In *Non-Monotonic Reasoning: 2nd International Workshop Proceedings*, pp. 74–99.

Gelfond, M. (2008). Answer sets. In van Harmelan, F., Lifschitz, V., and Porter, B. (Eds.), *Handbook of Knowledge Representation*, pp. 285–316. Elsevier.

Gelly, S. and Silver, D. (2008). Achieving master level play in 9 x 9 computer go. In *AAAI-08*, pp. 1537–1540.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. (1995). *Bayesian Data Analysis*. Chapman & Hall.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images. *PAMI*, *6*(6), 721–741.

Genesereth, M. R. (1984). The use of design descriptions in automated diagnosis. *AIJ*, *24*(1–3), 411–436.

Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann.

Genesereth, M. R. and Nourbakhsh, I. (1993). Time-saving tips for problem solving with incomplete information. In *AAAI-93*, pp. 724–730.

Genesereth, M. R. and Smith, D. E. (1981). Meta-level architecture. Memo HPP-81-6, Computer Science Department, Stanford University.

Gent, I., Petrie, K., and Puget, J.-F. (2006). Symmetry in constraint programming. In Rossi, F., van Beek, P., and Walsh, T. (Eds.), *Handbook of Constraint Programming*. Elsevier.

Gentner, D. (1983). Structure mapping: A theoretical framework for analogy. *Cognitive Science*, *7*, 155–170.

Gentner, D. and Goldin-Meadow, S. (Eds.). (2003). *Language in mind: Advances in the study of language and though*. MIT Press.

Gerevini, A. and Long, D. (2005). Plan constraints and preferences in PDDL3. Tech. rep., Dept. of Electronics for Automation, University of Brescia, Italy.

Gerevini, A. and Serina, I. (2002). LPG: A planner based on planning graphs with action costs. In *ICAPS-02*, pp. 281–290.

Gerevini, A. and Serina, I. (2003). Planning as propositional CSP: from walksat to local search for action graphs. *Constraints*, *8*, 389–413.

Gershwin, G. (1937). Let's call the whole thing off. Song.

Getoor, L. and Taskar, B. (Eds.). (2007). *Introduction to Statistical Relational Learning*. MIT Press.

Ghahramani, Z. and Jordan, M. I. (1997). Factorial hidden Markov models. *Machine Learning*, *29*, 245–274.

Ghahramani, Z. (1998). Learning dynamic bayesian networks. In *Adaptive Processing of Sequences and Data Structures*, pp. 168–197.

Ghahramani, Z. (2005). Tutorial on nonparametric Bayesian methods. Tutorial presentation at the

UAI Conference.

Ghallab, M., Howe, A., Knoblock, C. A., and McDermott, D. (1998). PDDL—The planning domain definition language. Tech. rep. DCS TR-1165, Yale Center for Computational Vision and Control.

Ghallab, M. and Laruelle, H. (1994). Representation and control in IxTeT, a temporal planner. In *AIPS-94*, pp. 61–67.

Ghallab, M., Nau, D. S., and Traverso, P. (2004). *Automated Planning: Theory and practice*. Morgan Kaufmann.

Gibbs, R. W. (2006). Metaphor interpretation as embodied simulation. *Mind*, *21*(3), 434–458.

Gibson, J. J. (1950). *The Perception of the Visual World*. Houghton Mifflin.

Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Houghton Mifflin.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (Eds.). (1996). *Markov chain Monte Carlo in practice*. Chapman and Hall.

Gilks, W. R., Thomas, A., and Spiegelhalter, D. J. (1994). A language and program for complex Bayesian modelling. *The Statistician*, *43*, 169–178.

Gilmore, P. C. (1960). A proof method for quantification theory: Its justification and realization. *IBM Journal of Research and Development*, *4*, 28–35.

Ginsberg, M. L. (1993). *Essentials of Artificial Intelligence*. Morgan Kaufmann.

Ginsberg, M. L. (1999). GIB: Steps toward an expert-level bridge-playing program. In *IJCAI-99*, pp. 584–589.

Ginsberg, M. L., Frank, M., Halpin, M. P., and Torrance, M. C. (1990). Search lessons learned from crossword puzzles. In *AAAI-90*, Vol. 1, pp. 210–215.

Ginsberg, M. L. (2001). GIB: Imperfect infoormation in a computationally challenging game. *JAIR*, *14*, 303–358.

Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions vis hashing. In *Proc. 25th Very Large Database (VLDB) Conference*.

Gittins, J. C. (1989). *Multi-Armed Bandit Allocation Indices*. Wiley.

Glanc, A. (1978). On the etymology of the word “robot”. *SIGART Newsletter*, *67*, 12.

Glover, F. and Laguna, M. (Eds.). (1997). *Tabu search*. Kluwer.

Gödel, K. (1930). *Uber die Vollständigkeit des Logikkalküls*. Ph.D. thesis, University of Vienna.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, *38*, 173–198.

Goebel, J., Volk, K., Walker, H., and Gerbault, F. (1989). Automatic classification of spectra from the infrared astronomical satellite (IRAS). *Astronomy and Astrophysics*, *222*, L5–L8.

Goertzel, B. and Pennachin, C. (2007). *Artificial General Intelligence*. Springer.

Gold, B. and Morgan, N. (2000). *Speech and Audio Signal Processing*. Wiley.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, *10*, 447–474.

Goldberg, A. V., Kaplan, H., and Werneck, R. F. (2006). Reach for a*: Efficient point-to-point shortest path algorithms. In *Workshop on algorithm engineering and experiments*, pp. 129–143.

Goldman, R. and Boddy, M. (1996). Expressive planning and explicit knowledge. In *AIPS-96*, pp. 110–117.

Goldszmidt, M. and Pearl, J. (1996). Qualitative probabilities for default reasoning, belief revision, and causal modeling. *AIJ*, *84*(1–2), 57–112.

Golomb, S. and Baumert, L. (1965). Backtrack pro-ramming. *JACM*, *14*, 516–524.

Golub, G., Heath, M., and Wahba, G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, *21*(2).

Gomes, C., Selman, B., Crato, N., and Kautz, H. (2000). Heavy-tailed phenomena in satisfiability and constrain processing. *JAR*, *24*, 67–100.

Gomes, C., Kautz, H., Sabharwal, A., and Selman, B. (2008). Satisfiability solvers. In van Harmelen, F., Lifschitz, V., and Porter, B. (Eds.), *Handbook of Knowledge Representation*. Elsevier.

Gomes, C. and Selman, B. (2001). Algorithm portfolios. *AIJ*, *126*, 43–62.

Gomes, C., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *AAAI-98*, pp. 431–437.

Gonthier, G. (2008). Formal proof–The four-color theorem. *Notices of the AMS*, *55*(11), 1382–1393.

Good, I. J. (1961). A causal calculus. *British Journal of the Philosophy of Science*, *11*, 305–318.

Good, I. J. (1965). Speculations concerning the first ultraintelligent machine. In Alt, F. L. and Rubinoff, M. (Eds.), *Advances in Computers*, Vol. 6, pp. 31–88. Academic Press.

Good, I. J. (1983). *Good Thinking: The Foundations of Probability and Its Applications*. University of Minnesota Press.

Goodman, D. and Keene, R. (1997). *Man versus Machine: Kasparov versus Deep Blue*. H3 Publications.

Goodman, J. (2001). A bit of progress in language modeling. Tech. rep. MSR-TR-2001-72, Microsoft Research.

Goodman, J. and Heckerman, D. (2004). Fighting spam with statistics. *Significance, the Magazine of the Royal Statistical Society*, *1*, 69–72.

Goodman, N. (1954). *Fact, Fiction and Forecast*. University of London Press.

Goodman, N. (1977). *The Structure of Appearance* (3rd edition). D. Reidel.

Gopnik, A. and Glymour, C. (2002). Causal maps and bayes nets: A cognitive and computational account of theory-formation. In Caruthers, P., Stich, S., and Siegal, M. (Eds.), *The Cognitive Basis of*

Science. Cambridge University Press.

Gordon, D. M. (2000). *Ants at Work*. Norton.

Gordon, D. M. (2007). Control without hierarchy. *Nature*, *446*(8), 143.

Gordon, M. J., Milner, A. J., and Wadsworth, C. P. (1979). *Edinburgh LCF*. Springer-Verlag.

Gordon, N. (1994). *Bayesian methods for tracking*. Ph.D. thesis, Imperial College.

Gordon, N., Salmond, D. J., and Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, *140*(2), 107–113.

Gorry, G. A. (1968). Strategies for computer-aided diagnosis. *Mathematical Biosciences*, *2*(3–4), 293–318.

Gorry, G. A., Kassirer, J. P., Essig, A., and Schwartz, W. B. (1973). Decision analysis as the basis for computer-aided management of acute renal failure. *American Journal of Medicine*, *55*, 473–484.

Gottlob, G., Leone, N., and Scarcello, F. (1999a). A comparison of structural CSP decomposition methods. In *IJCAI-99*, pp. 394–399.

Gottlob, G., Leone, N., and Scarcello, F. (1999b). Hypertree decompositions and tractable queries. In *PODS-99*, pp. 21–32.

Graham, S. L., Harrison, M. A., and Ruzzo, W. L. (1980). An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, *2*(3), 415–462.

Grama, A. and Kumar, V. (1995). A survey of parallel search algorithms for discrete optimization problems. *ORSA Journal of Computing*, *7*(4), 365–385.

Grassmann, H. (1861). *Lehrbuch der Arithmetik*. Th. Chr. Fr. Enslin, Berlin.

Grayson, C. J. (1960). Decisions under uncertainty: Drilling decisions by oil and gas operators. Tech. rep., Division of Research, Harvard Business School.

Green, B., Wolf, A., Chomsky, C., and Laugherty, K. (1961). BASEBALL: An automatic question answerer. In *Proc. Western Joint Computer Conference*, pp. 219–224.

Green, C. (1969a). Application of theorem proving to problem solving. In *IJCAI-69*, pp. 219–239.

Green, C. (1969b). Theorem-proving by resolution as a basis for question-answering systems. In Meltzer, B., Michie, D., and Swann, M. (Eds.), *Machine Intelligence 4*, pp. 183–205. Edinburgh University Press.

Green, C. and Raphael, B. (1968). The use of theorem-proving techniques in question-answering systems. In *Proc. 23rd ACM National Conference*.

Greenblatt, R. D., Eastlake, D. E., and Crocker, S. D. (1967). The Greenblatt chess program. In *Proc. Fall Joint Computer Conference*, pp. 801–810.

Greiner, R. (1989). Towards a formal analysis of EBL. In *ICML-89*, pp. 450–453.

Grinstead, C. and Snell, J. (1997). *Introduction to Probability*. AMS.

Grove, W. and Meehl, P. (1996). Comparative efficiency of informal (subjective, impressionistic) and formal (mechanical, algorithmic) prediction procedures: The clinical statistical controversy. *Psychology, Public Policy, and Law*, *2*, 293–323.

Gruber, T. (2004). Interview of Tom Gruber. *AIS SIGSEMIS Bulletin*, *1*(3).

Gu, J. (1989). *Parallel Algorithms and Architectures for Very Fast AI Search*. Ph.D. thesis, University of Utah.

Guard, J., Oglesby, F., Bennett, J., and Settle, L. (1969). Semi-automated mathematics. *JACM*, *16*, 49–62.

Guestrin, C., Koller, D., Gearhart, C., and Kanodia, N. (2003a). Generalizing plans to new environments in relational MDPs. In *IJCAI-03*.

Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. (2003b). Efficient solution algorithms for factored MDPs. *JAIR*, *19*, 399–468.

Guestrin, C., Lagoudakis, M. G., and Parr, R. (2002). Coordinated reinforcement learning. In *ICML-02*, pp. 227–234.

Guibas, L. J., Knuth, D. E., and Sharir, M. (1992). Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, *7*, 381–413. See also *17th Int. Coll. on Automata, Languages and Programming*, 1990, pp. 414–431.

Gumperz, J. and Levinson, S. (1996). *Rethinking Linguistic Relativity*. Cambridge University Press.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *JMLR*, pp. 1157– 1182.

Hacking, I. (1975). *The Emergence of Probability*. Cambridge University Press.

Haghighi, A. and Klein, D. (2006). Prototype-driven grammar induction. In *COLING-06*.

Hald, A. (1990). *A History of Probability and Statistics and Their Applications before 1750*. Wiley.

Halevy, A. (2007). Dataspaces: A new paradigm for data integration. In *Brazilian Symposium on Databases*.

Halevy, A., Norvig, P., and Pereira, F. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems*, *March/April*, 8–12.

Halpern, J. Y. (1990). An analysis of first-order logics of probability. *AIJ*, *46*(3), 311–350.

Halpern, J. Y. (1999). Technical addendum, Cox's theorem revisited. *JAIR*, *11*, 429–435.

Halpern, J. Y. and Weissman, V. (2008). Using firstorder logic to reason about policies. *ACM Transactions on Information and System Security*, *11*(4).

Hamming, R. W. (1991). *The Art of Probability for Scientists and Engineers*. Addison-Wesley.

Hammond, K. (1989). *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press.

Hamscher, W., Console, L., and Kleer, J. D. (1992). *Readings in Model-based Diagnosis*. Morgan Kaufmann.

Han, X. and Boyden, E. (2007). Multiple-color optical activation, silencing, and desynchronization of neural activity, with single-spike temporal resolution. *PLoS One*, *e299*.

Hand, D., Mannila, H., and Smyth, P. (2001). *Principles of Data Mining*. MIT Press.

Handschin, J. E. and Mayne, D. Q. (1969). Monte Carlo techniques to estimate the conditional expectation in multi-stage nonlinear filtering. *Int. J. Control*, *9*(5), 547–559.

Hansen, E. (1998). Solving POMDPs by searching in policy space. In *UAI-98*, pp. 211–219.

Hansen, E. and Zilberstein, S. (2001). LAO*: a heuristic search algorithm that finds solutions with loops. *AIJ*, *129*(1–2), 35–62.

Hansen, P. and Jaumard, B. (1990). Algorithms for the maximum satisfiability problem. *Computing*, *44*(4), 279–303.

Hanski, I. and Cambefort, Y. (Eds.). (1991). *Dung Beetle Ecology*. Princeton University Press.

Hansson, O. and Mayer, A. (1989). Heuristic search as evidential reasoning. In *UAI 5*.

Hansson, O., Mayer, A., and Yung, M. (1992). Criticizing solutions to relaxed models yields powerful admissible heuristics. *Information Sciences*, *63*(3), 207–227.

Haralick, R. M. and Elliot, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *AIJ*, *14*(3), 263–313.

Hardin, G. (1968). The tragedy of the commons. *Science*, *162*, 1243–1248.

Hardy, G. H. (1940). *A Mathematician's Apology*. Cambridge University Press.

Harman, G. H. (1983). *Change in View: Principles of Reasoning*. MIT Press.

Harris, Z. (1954). Distributional structure. *Word*, *10*(2/3).

Harrison, J. R. and March, J. G. (1984). Decision making and postdecision surprises. *Administrative Science Quarterly*, *29*, 26–42.

Harsanyi, J. (1967). Games with incomplete information played by Bayesian players. *Management Science*, *14*, 159–182.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *SSC-4(2)*, 100–107.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1972). Correction to "A formal basis for the heuristic determination of minimum cost paths". *SIGART Newsletter*, *37*, 28–29.

Hart, T. P. and Edwards, D. J. (1961). The tree prune (TP) algorithm. Artificial intelligence project memo 30, Massachusetts Institute of Technology.

Hartley, H. (1958). Maximum likelihood estimation from incomplete data. *Biometrics*, *14*, 174–194.

Hartley, R. and Zisserman, A. (2000). *Multiple view geometry in computer vision*. Cambridge

University Press.

Haslum, P., Botea, A., Helmert, M., Bonet, B., and Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI-07*, pp. 1007–1012.

Haslum, P. and Geffner, H. (2001). Heuristic planning with time and resources. In *Proc. IJCAI-01 Workshop on Planning with Resources*.

Haslum, P. (2006). Improving heuristics through relaxed search – An analysis of TP4 and HSP*a in the 2004 planning competition. *JAIR*, *25*, 233–267.

Haslum, P., Bonet, B., and Geffner, H. (2005). New admissible heuristics for domain-independent planning. In *AAAI-05*.

Hastie, T. and Tibshirani, R. (1996). Discriminant adaptive nearest neighbor classification and regression. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E. (Eds.), *NIPS 8*, pp. 409–15. MIT Press.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference and Prediction* (2nd edition). Springer-Verlag.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction* (2nd edition). Springer-Verlag.

Haugeland, J. (Ed.). (1985). *Artificial Intelligence: The Very Idea*. MIT Press.

Hauk, T. (2004). *Search in Trees with Chance Nodes*. Ph.D. thesis, Univ. of Alberta.

Haussler, D. (1989). Learning conjunctive concepts in structural domains. *Machine Learning*, *4*(1), 7–40.

Havelund, K., Lowry, M., Park, S., Pecheur, C., Penix, J., Visser, W., and White, J. L. (2000). Formal analysis of the remote agent before and after flight. In *Proc. 5th NASA Langley Formal Methods Workshop*.

Havenstein, H. (2005). Spring comes to AI winter. *Computer World*.

Hawkins, J. and Blakeslee, S. (2004). *On Intelligence*. Henry Holt and Co.

Hayes, P. J. (1978). The naive physics manifesto. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age*. Edinburgh University Press.

Hayes, P. J. (1979). The logic of frames. In Metzing, D. (Ed.), *Frame Conceptions and Text Understanding*, pp. 46–61. de Gruyter.

Hayes, P. J. (1985a). Naive physics I: Ontology for liquids. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 3, pp. 71–107. Ablex.

Hayes, P. J. (1985b). The second naive physics manifesto. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 1, pp. 1–36. Ablex.

Haykin, S. (2008). *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

Hays, J. and Efros, A. A. (2007). Scene completion Using millions of photographs. *ACM*

Transactions on Graphics (SIGGRAPH), *26*(3).

Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *COLING-92*.

Hearst, M. A. (2009). *Search User Interfaces*. Cambridge University Press.

Hebb, D. O. (1949). *The Organization of Behavior*. Wiley.

Heckerman, D. (1986). Probabilistic interpretation for MYCIN's certainty factors. In Kanal, L. N. and Lemmer, J. F. (Eds.), *UAI 2*, pp. 167–196. Elsevier/North-Holland.

Heckerman, D. (1991). *Probabilistic Similarity Networks*. MIT Press.

Heckerman, D. (1998). A tutorial on learning with Bayesian networks. In Jordan, M. I. (Ed.), *Learning in graphical models*. Kluwer.

Heckerman, D., Geiger, D., and Chickering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. Technical report MSR-TR-94-09, Microsoft Research.

Heidegger, M. (1927). *Being and Time*. SCM Press.

Heinz, E. A. (2000). *Scalable search in computer chess*. Vieweg.

Held, M. and Karp, R. M. (1970). The traveling salesman problem and minimum spanning trees. *Operations Research*, *18*, 1138–1162.

Helmert, M. (2001). On the complexity of planning in transportation domains. In *ECP-01*.

Helmert, M. (2003). Complexity results for standard benchmark domains in planning. *AIJ*, *143*(2), 219–262.

Helmert, M. (2006). The fast downward planning system. *JAIR*, *26*, 191–246.

Helmert, M. and Richter, S. (2004). Fast downward – Making use of causal dependencies in the problem representation. In *Proc. International Planning Competition at ICAPS*, pp. 41–43.

Helmert, M. and R¨oger, G. (2008). How good is almost perfect? In *AAAI-08*.

Hendler, J., Carbonell, J. G., Lenat, D. B., Mizoguchi, R., and Rosenbloom, P. S. (1995). VERY large knowledge bases – Architecture vs engineering. In *IJCAI-95*, pp. 2033–2036.

Henrion, M. (1988). Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F. and Kanal, L. N. (Eds.), *UAI 2*, pp. 149–163. Elsevier/North-Holland.

Henzinger, T. A. and Sastry, S. (Eds.). (1998). *Hybrid systems: Computation and control*. Springer-Verlag.

Herbrand, J. (1930). *Recherches sur la Th´eorie de la D´emonstration*. Ph.D. thesis, University of Paris.

Hewitt, C. (1969). PLANNER: a language for proving theorems in robots. In *IJCAI-69*, pp. 295–301.

Hierholzer, C. (1873). Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, *6*, 30–32.

Hilgard, E. R. and Bower, G. H. (1975). *Theories of Learning* (4th edition). Prentice-Hall.

Hintikka, J. (1962). *Knowledge and Belief*. Cornell University Press.

Hinton, G. E. and Anderson, J. A. (1981). *Parallel Models of Associative Memory*. Lawrence Erlbaum Associates.

Hinton, G. E. and Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, *1*(3), 495–502.

Hinton, G. E., Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–15554.

Hinton, G. E. and Sejnowski, T. (1983). Optimal perceptual inference. In *CVPR*, pp. 448–453.

Hinton, G. E. and Sejnowski, T. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing*, chap. 7, pp. 282–317. MIT Press.

Hirsh, H. (1987). Explanation-based generalization in a logic programming environment. In *IJCAI-87*.

Hobbs, J. R. (1990). *Literature and Cognition*. CSLI Press.

Hobbs, J. R., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M. E., and Tyson, M. (1997). FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In Roche, E. and Schabes, Y. (Eds.), *Finite-State Devices for Natural Language Processing*, pp. 383–406. MIT Press.

Hobbs, J. R. and Moore, R. C. (Eds.). (1985). *Formal Theories of the Commonsense World*. Ablex.

Hobbs, J. R., Stickel, M. E., Appelt, D., and Martin, P. (1993). Interpretation as abduction. *AIJ*, *63*(1–2), 69–142.

Hoffmann, J. (2001). FF: The fast-forward planning system. *AIMag*, *22*(3), 57–62.

Hoffmann, J. and Brafman, R. I. (2006). Conformant planning via heuristic forward search: A new approach. *AIJ*, *170*(6–7), 507–541.

Hoffmann, J. and Brafman, R. I. (2005). Contingent planning via heuristic forward search with implicit belief states. In *ICAPS-05*.

Hoffmann, J. (2005). Where “ignoring delete lists” works: Local search topology in planning benchmarks. *JAIR*, *24*, 685–758.

Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *JAIR*, *14*, 253–302.

Hoffmann, J., Sabharwal, A., and Domshlak, C. (2006). Friends or foes? An AI planning perspective on abstraction and search. In *ICAPS-06*, pp. 294–303.

Hogan, N. (1985). Impedance control: An approach to manipulation. Parts I, II, and III. *J. Dynamic Systems, Measurement, and Control*, *107*(3), 1–24.

Hoiem, D., Efros, A. A., and Hebert, M. (2008). Putting objects in perspective. *IJCV*, *80*(1).

Holland, J. H. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press.

Holland, J. H. (1995). *Hidden Order: How Adaptation Builds Complexity*. Addison-Wesley.

Holte, R. and Hernadvolgyi, I. (2001). Steps towards the automatic creation of search heuristics. Tech. rep. TR04-02, CS Dept., Univ. of Alberta.

Holzmann, G. J. (1997). The Spin model checker. *IEEE Transactions on Software Engineering*, *23*(5), 279–295.

Hood, A. (1824). Case 4th—28 July 1824 (Mr. Hood's cases of injuries of the brain). *Phrenological Journal and Miscellany*, *2*, 82–94.

Hooker, J. (1995). Testing heuristics: We have it all wrong. *J. Heuristics*, *1*, 33–42.

Hoos, H. and Tsang, E. (2006). Local search methods. In Rossi, F., van Beek, P., and Walsh, T. (Eds.), *Handbook of Constraint Processing*, pp. 135–168. Elsevier.

Hope, J. (1994). *The Authorship of Shakespeare's Plays*. Cambridge University Press.

Hopfield, J. J. (1982). Neurons with graded response have collective computational properties like those of two-state neurons. *PNAS*, *79*, 2554–2558.

Horn, A. (1951). On sentences which are true of direct unions of algebras. *JSL*, *16*, 14–21.

Horn, B. K. P. (1970). Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. Technical report 232, MIT Artificial Intelligence Laboratory.

Horn, B. K. P. (1986). *Robot Vision*. MIT Press.

Horn, B. K. P. and Brooks, M. J. (1989). *Shape from Shading*. MIT Press.

Horn, K. V. (2003). Constructing a logic of plausible inference: A guide to cox's theorem. *IJAR*, *34*, 3–24.

Horning, J. J. (1969). *A study of grammatical inference*. Ph.D. thesis, Stanford University.

Horowitz, E. and Sahni, S. (1978). *Fundamentals of Computer Algorithms*. Computer Science Press.

Horswill, I. (2000). Functional programming of behavior-based systems. *Autonomous Robots*, *9*, 83–93.

Horvitz, E. J. (1987). Problem-solving design: Reasoning about computational value, trade-offs, and resources. In *Proc. Second Annual NASA Research Forum*, pp. 26–43.

Horvitz, E. J. (1989). Rational metareasoning and compilation for optimizing decisions under bounded resources. In *Proc. Computational Intelligence 89*. Association for Computing Machinery.

Horvitz, E. J. and Barry, M. (1995). Display of information for time-critical decision making. In *UAI95*, pp. 296–305.

Horvitz, E. J., Breese, J. S., Heckerman, D., and Hovel, D. (1998). The Lumiere project: Bayesian

user modeling for inferring the goals and needs of software users. In *UAI-98*, pp. 256–265.

Horvitz, E. J., Breese, J. S., and Henrion, M. (1988). Decision theory in expert systems and artificial intelligence. *IJAR*, *2*, 247–302.

Horvitz, E. J. and Breese, J. S. (1996). Ideal partition of resources for metareasoning. In *AAAI-96*, pp. 1229–1234.

Horvitz, E. J. and Heckerman, D. (1986). The inconsistent use of measures of certainty in artificial intelligence research. In Kanal, L. N. and Lemmer, J. F. (Eds.), *UAI 2*, pp. 137–151. Elsevier/North-Holland.

Horvitz, E. J., Heckerman, D., and Langlotz, C. P. (1986). A framework for comparing alternative formalisms for plausible reasoning. In *AAAI-86*, Vol. 1, pp. 210–214.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press.

Howard, R. A. (1966). Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, *SSC-2*, 22–26.

Howard, R. A. (1977). Risk preference. In Howard, R. A. and Matheson, J. E. (Eds.), *Readings in Decision Analysis*, pp. 429–465. Decision Analysis Group, SRI International.

Howard, R. A. (1989). Microrisks for medical decision analysis. *Int. J. Technology Assessment in Health Care*, *5*, 357–370.

Howard, R. A. and Matheson, J. E. (1984). Influence diagrams. In Howard, R. A. and Matheson, J. E. (Eds.), *Readings on the Principles and Applications of Decision Analysis*, pp. 721–762. Strategic Decisions Group.

Howe, D. (1987). The computational behaviour of girard's paradox. In *LICS-87*, pp. 205–214.

Hsu, F.-H. (2004). *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press.

Hsu, F.-H., Anantharaman, T. S., Campbell, M. S., and Nowatzyk, A. (1990). A grandmaster chess machine. *Scientific American*, *263*(4), 44–50.

Hu, J. and Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *ICML-98*, pp. 242–250.

Hu, J. and Wellman, M. P. (2003). Nash q-learning for general-sum stochastic games. *JMLR*, *4*, 1039– 1069.

Huang, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S. J., and Weber, J. (1994). Automatic symbolic traffic scene analysis using belief networks. In *AAAI-94*, pp. 966–972.

Huang, T. and Russell, S. J. (1998). Object identification: A Bayesian analysis with application to traffic surveillance. *AIJ*, *103*, 1–17.

Huang, X. D., Acero, A., and Hon, H. (2001). *Spoken Language Processing*. Prentice Hall.

Hubel, D. H. (1988). *Eye, Brain, and Vision*. W. H. Freeman.

Huddleston, R. D. and Pullum, G. K. (2002). *The Cambridge Grammar of the English Language*. Cambridge University Press.

Huffman, D. A. (1971). Impossible objects as nonsense sentences. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 6*, pp. 295–324. Edinburgh University Press.

Hughes, B. D. (1995). *Random Walks and Random Environments, Vol. 1: Random Walks*. Oxford University Press.

Hughes, G. E. and Cresswell, M. J. (1996). *A New Introduction to Modal Logic*. Routledge.

Huhns, M. N. and Singh, M. P. (Eds.). (1998). *Readings in Agents*. Morgan Kaufmann.

Hume, D. (1739). *A Treatise of Human Nature* (2nd edition). Republished by Oxford University Press, 1978, Oxford, UK.

Humphrys, M. (2008). How my program passed the turing test. In Epstein, R., Roberts, G., and Beber, G. (Eds.), *Parsing the Turing Test*. Springer.

Hunsberger, L. and Grosz, B. J. (2000). A combinatorial auction for collaborative planning. In *Int. Conference on Multi-Agent Systems (ICMAS-2000)*.

Hunt, W. and Brock, B. (1992). A formal HDL and its use in the FM9001 verification. *Philosophical Transactions of the Royal Society of London*, *339*.

Hunter, L. and States, D. J. (1992). Bayesian classification of protein structure. *IEEE Expert*, *7*(4), 67–75.

Hurst, M. (2000). *The Interpretation of Text in Tables*. Ph.D. thesis, Edinburgh.

Hurwicz, L. (1973). The design of mechanisms for resource allocation. *American Economic Review Papers and Proceedings*, *63*(1), 1–30.

Husmeier, D. (2003). Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic bayesian networks. *Bioinformatics*, *19*(17), 2271–2282.

Huth, M. and Ryan, M. (2004). *Logic in computer science: modelling and reasoning about systems* (2nd edition). Cambridge University Press.

Huttenlocher, D. and Ullman, S. (1990). Recognizing solid objects by alignment with an image. *IJCV*, *5*(2), 195–212.

Huygens, C. (1657). De ratiociniis in ludo aleae. In van Schooten, F. (Ed.), *Exercitionum Mathematicorum*. Elsevirii, Amsterdam. Translated into English by John Arbuthnot (1692).

Huyn, N., Dechter, R., and Pearl, J. (1980). Probabilistic analysis of the complexity of A*. *AIJ*, *15*(3), 241–254.

Hwa, R. (1998). An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *ACL98*, pp. 557–563.

Hwang, C. H. and Schubert, L. K. (1993). EL: A formal, yet natural, comprehensive knowledge representation. In *AAAI-93*, pp. 676–682.

Ingerman, P. Z. (1967). Panini–Backus form suggested. *CACM*, *10*(3), 137.

Inoue, K. (2001). Inverse entailment for full clausal theories. In *LICS-2001 Workshop on Logic and Learning*.

Intille, S. and Bobick, A. (1999). A framework for recognizing multi-agent action from visual evidence. In *AAAI-99*, pp. 518–525.

Isard, M. and Blake, A. (1996). Contour tracking by stochastic propagation of conditional density. In *ECCV*, pp. 343–356.

Iwama, K. and Tamaki, S. (2004). Improved upper bounds for 3-SAT. In *SODA-04*.

Jaakkola, T. and Jordan, M. I. (1996). Computing upper and lower bounds on likelihoods in intractable networks. In *UAI-96*, pp. 340–348. Morgan Kaufmann.

Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In *NIPS 7*, pp. 345–352.

Jackson, F. (1982). Epiphenomenal qualia. *Philosophical Quarterly*, *32*, 127–136.

Jaffar, J. and Lassez, J.-L. (1987). Constraint logic programming. In *Proc. Fourteenth ACM Conference on Principles of Programming Languages*, pp. 111–119. Association for Computing Machinery.

Jaffar, J., Michaylov, S., Stuckey, P. J., and Yap, R. H. C. (1992). The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems*, *14*(3), 339–395.

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge Univ. Press.

Jefferson, G. (1949). The mind of mechanical man: The Lister Oration delivered at the Royal College of Surgeons in England. *British Medical Journal*, *1*(25), 1105–1121.

Jeffrey, R. C. (1983). *The Logic of Decision* (2nd edition). University of Chicago Press.

Jeffreys, H. (1948). *Theory of Probability*. Oxford.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proc. IEEE*, *64*(4), 532–556.

Jelinek, F. (1997). *Statistical Methods for Speech Recognition*. MIT Press.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice*, pp. 381–397.

Jennings, H. S. (1906). *Behavior of the Lower Organisms*. Columbia University Press.

Jenniskens, P., Betlem, H., Betlem, J., and Barifaijo, E. (1994). The Mbale meteorite shower. *Meteoritics*, *29*(2), 246–254.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. Springer-Verlag.

Jensen, F. V. (2007). *Bayesian Networks and Decision Graphs*. Springer-Verlag.

Jevons, W. S. (1874). *The Principles of Science*. Routledge/Thoemmes Press, London.

Ji, S., Parr, R., Li, H., Liao, X., and Carin, L. (2007). Point-based policy iteration. In *AAAI-07*.

Jimenez, P. and Torras, C. (2000). An efficient algorithm for searching implicit AND/OR graphs with cycles. *AIJ*, *124*(1), 1–30.

Joachims, T. (2001). A statistical learning model of text classification with support vector machines. In *SIGIR-01*, pp. 128–136.

Johnson, W. W. and Story, W. E. (1879). Notes on the "15" puzzle. *American Journal of Mathematics*, *2*, 397–404.

Johnston, M. D. and Adorf, H.-M. (1992). Scheduling with neural networks: The case of the Hubble space telescope. *Computers and Operations Research*, *19*(3–4), 209–240.

Jones, N. D., Gomard, C. K., and Sestoft, P. (1993). *Partial Evaluation and Automatic Program Generation*. Prentice-Hall.

Jones, R., Laird, J., and Nielsen, P. E. (1998). Automated intelligent pilots for combat flight simulation. In *AAAI-98*, pp. 1047–54.

Jones, R., McCallum, A., Nigam, K., and Riloff, E. (1999). Bootstrapping for text learning tasks. In *Proc. IJCAI-99 Workshop on Text Mining: Foundations, Techniques, and Applications*, pp. 52–63.

Jones, T. (2007). *Artificial Intelligence: A Systems Approach*. Infinity Science Press.

Jonsson, A., Morris, P., Muscettola, N., Rajan, K., and Smith, B. (2000). Planning in interplanetary space: Theory and practice. In *AIPS-00*, pp. 177–186.

Jordan, M. I. (1995). Why the logistic function? a tutorial discussion on probabilities and neural networks. Computational cognitive science technical report 9503, Massachusetts Institute of Technology.

Jordan, M. I. (2005). Dirichlet processes, Chinese restaurant processes and all that. Tutorial presentation at the NIPS Conference.

Jordan, M. I., Ghahramani, Z., Jaakkola, T., and Saul, L. K. (1998). An introduction to variational methods for graphical models. In Jordan, M. I. (Ed.), *Learning in Graphical Models*. Kluwer.

Jouannaud, J.-P. and Kirchner, C. (1991). Solving equations in abstract algebras: A rule-based survey of unification. In Lassez, J.-L. and Plotkin, G. (Eds.), *Computational Logic*, pp. 257–321. MIT Press.

Judd, J. S. (1990). *Neural Network Design and the Complexity of Learning*. MIT Press.

Juels, A. and Wattenberg, M. (1996). Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E. (Eds.), *NIPS 8*, pp. 430–6. MIT Press.

Junker, U. (2003). The logic of ilog (j)configurator: Combining constraint programming with a description logic. In *Proc. IJCAI-03 Configuration Workshop*, pp. 13–20.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall.

Jurafsky, D. and Martin, J. H. (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (2nd edition). Prentice-Hall.

Kadane, J. B. and Simon, H. A. (1977). Optimal strategies for a class of constrained sequential problems. *Annals of Statistics*, *5*, 237–255.

Kadane, J. B. and Larkey, P. D. (1982). Subjective probability and the theory of games. *Management Science*, *28*(2), 113–120.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and actiong in partially observable stochastic domains. *AIJ*, *101*, 99–134.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *JAIR*, *4*, 237–285.

Kaelbling, L. P. and Rosenschein, S. J. (1990). Action and planning in embedded agents. *Robotics and Autonomous Systems*, *6*(1–2), 35–48.

Kager, R. (1999). *Optimality Theory*. Cambridge University Press.

Kahn, H. and Marshall, A. W. (1953). Methods of reducing sample size in Monte Carlo computations. *Operations Research*, *1*(5), 263–278.

Kahneman, D., Slovic, P., and Tversky, A. (Eds.). (1982). *Judgment under Uncertainty: Heuristics and Biases*. Cambridge University Press.

Kahneman, D. and Tversky, A. (1979). Prospect theory: An analysis of decision under risk. *Econometrica*, pp. 263–291.

Kaindl, H. and Khorsand, A. (1994). Memory-bounded bidirectional search. In *AAAI-94*, pp. 1359–1364.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *J. Basic Engineering*, *82*, 35–46.

Kambhampati, S. (1994). Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence*, *10*, 213–244.

Kambhampati, S., Mali, A. D., and Srivastava, B. (1998). Hybrid planning for partially hierarchical domains. In *AAAI-98*, pp. 882–888.

Kanal, L. N. and Kumar, V. (1988). *Search in Artificial Intelligence*. Springer-Verlag.

Kanazawa, K., Koller, D., and Russell, S. J. (1995). Stochastic simulation algorithms for dynamic probabilistic networks. In *UAI-95*, pp. 346–351.

Kantorovich, L. V. (1939). Mathematical methods of organizing and planning production. Publishd in translation in *Management Science*, *6*(4), 366–422, July 1960.

Kaplan, D. and Montague, R. (1960). A paradox regained. *Notre Dame Journal of Formal Logic*, *1*(3), 79–90.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*,

4, 373–395.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W. (Eds.), *Complexity of Computer Computations*, pp. 85–103. Plenum.

Kartam, N. A. and Levitt, R. E. (1990). A constraint-based approach to construction planning of multi-story buildings. In *Expert Planning Systems*, pp. 245–250. Institute of Electrical Engineers.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Tech. rep. AFCRL-65-758, Air Force Cambridge Research Laboratory.

Kasparov, G. (1997). IBM owes me a rematch. *Time*, *149*(21), 66–67.

Kaufmann, M., Manolios, P., and Moore, J. S. (2000). *Computer-Aided Reasoning: An Approach*. Kluwer.

Kautz, H. (2006). Deconstructing planning as satisfiability. In *AAAI-06*.

Kautz, H., McAllester, D. A., and Selman, B. (1996). Encoding plans in propositional logic. In *KR-96*, pp. 374–384.

Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *ECAI-92*, pp. 359–363.

Kautz, H. and Selman, B. (1998). BLACKBOX: A new approach to the application of theorem proving to problem solving. Working Notes of the AIPS-98 Workshop on Planning as Combinatorial Search.

Kavraki, L., Svestka, P., Latombe, J.-C., and Over-mars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, *12*(4), 566–580.

Kay, M., Gawron, J. M., and Norvig, P. (1994). *Verbmobil: A Translation System for Face-To-Face Dialog*. CSLI Press.

Kearns, M. (1990). *The Computational Complexity of Machine Learning*. MIT Press.

Kearns, M., Mansour, Y., and Ng, A. Y. (2000). Approximate planning in large POMDPs via reusable trajectories. In Solla, S. A., Leen, T. K., and Müller, K.-R. (Eds.), *NIPS 12*. MIT Press.

Kearns, M. and Singh, S. P. (1998). Near-optimal reinforcement learning in polynomial time. In *ICML-98*, pp. 260–268.

Kearns, M. and Vazirani, U. (1994). *An Introduction to Computational Learning Theory*. MIT Press.

Kearns, M. and Mansour, Y. (1998). A fast, bottom-up decision tree pruning algorithm with near-optimal generalization. In *ICML-98*, pp. 269–277.

Kebeasy, R. M., Hussein, A. I., and Dahy, S. A. (1998). Discrimination between natural earthquakes and nuclear explosions using the Aswan Seismic Network. *Annali di Geofisica*, *41*(2), 127–140.

Keeney, R. L. (1974). Multiplicative utility functions. *Operations Research*, *22*, 22–34.

Keeney, R. L. and Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value*

Tradeoffs. Wiley.

Kemp, M. (Ed.). (1989). *Leonardo on Painting: An Anthology of Writings*. Yale University Press.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer*, *36*(1), 41–50.

Kersting, K., Raedt, L. D., and Kramer, S. (2000). Interpreting bayesian logic programs. In *Proc. AAAI2000 Workshop on Learning Statistical Models from Relational Data*.

Kessler, B., Nunberg, G., and Schütze, H. (1997). Automatic detection of text genre. *CoRR*, *cmplg/9707002*.

Keynes, J. M. (1921). *A Treatise on Probability*. Macmillan.

Khare, R. (2006). Microformats: The next (small) thing on the semantic web. *IEEE Internet Computing*, *10*(1), 68–75.

Khatib, O. (1986). Real-time obstacle avoidance for robot manipulator and mobile robots. *Int. J. Robotics Research*, *5*(1), 90–98.

Khmelev, D. V. and Tweedie, F. J. (2001). Using Markov chains for identification of writer. *Literary and Linguistic Computing*, *16*(3), 299–307.

Kietz, J.-U. and D**u**zeroski, S. (1994). Inductive logic programming and learnability. *SIGART Bulletin*, *5*(1), 22–32.

Kilgarriff, A. and Grefenstette, G. (2006). Introduction to the special issue on the web as corpus. *Computational Linguistics*, *29*(3), 333–347.

Kim, J. H. (1983). *CONVINCE: A Conversational Inference Consolidation Engine*. Ph.D. thesis, Department of Computer Science, University of California at Los Angeles.

Kim, J. H. and Pearl, J. (1983). A computational model for combined causal and diagnostic reasoning in inference systems. In *IJCAI-83*, pp. 190–193.

Kim, J.-H., Lee, C.-H., Lee, K.-H., and Kuppuswamy, N. (2007). Evolving personality of a genetic robot in ubiquitous environment. In *The 16th IEEE International Symposium on Robot and Human interactive Communication*, pp. 848–853.

King, R. D., Rowland, J., Oliver, S. G., and Young, M. (2009). The automation of science. *Science*, *324*(5923), 85–89.

Kirk, D. E. (2004). *Optimal Control Theory: An Introduction*. Dover.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Kister, J., Stein, P., Ulam, S., Walden, W., and Wells, M. (1957). Experiments in chess. *JACM*, *4*, 174–177.

Kisynski, J. and Poole, D. (2009). Lifted aggregation in directed first-order probabilistic models. In *IJCAI-09*.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1997a). RoboCup: The robot world cup initiative. In *Proc. First International Conference on Autonomous Agents*, pp. 340–347.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., and Matsubara, H. (1997b). RoboCup: A challenge problem for AI. *AIMag*, *18*(1), 73–85.

Kjaerulff, U. (1992). A computational scheme for reasoning in dynamic probabilistic networks. In *UAI-92*, pp. 121–129.

Klein, D. and Manning, C. (2001). Parsing with tree-bank grammars: Empirical bounds, theoretical models, and the structure of the Penn treebank. In *ACL 01*.

Klein, D. and Manning, C. (2003). A* parsing: Fast exact Viterbi parse selection. In *HLT-NAACL-03*, pp. 119–126.

Klein, D., Smarr, J., Nguyen, H., and Manning, C. (2003). Named entity recognition with character-level models. In *Conference on Natural Language Learning (CoNLL)*.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *JACM*, *46*(5), 604–632.

Klemperer, P. (2002). What really matters in auction design. *J. Economic Perspectives*, *16*(1).

Kneser, R. and Ney, H. (1995). Improved backing-off for M-gram language modeling. In *ICASSP-95*, pp. 181–184.

Knight, K. (1999). A statistical MT tutorial workbook. Prepared in connection with the Johns Hopkins University summer workshop.

Knuth, D. E. (1964). Representing numbers using only one 4. *Mathematics Magazine*, *37*(Nov/Dec), 308–310.

Knuth, D. E. (1968). Semantics for context-free languages. *Mathematical Systems Theory*, *2*(2), 127–145.

Knuth, D. E. (1973). *The Art of Computer Programming* (second edition)., Vol. 2: Fundamental Algorithms. Addison-Wesley.

Knuth, D. E. (1975). An analysis of alpha–beta pruning. *AIJ*, *6*(4), 293–326.

Knuth, D. E. and Bendix, P. B. (1970). Simple word problems in universal algebras. In Leech, J. (Ed.), *Computational Problems in Abstract Algebra*, pp. 263–267. Pergamon.

Kocsis, L. and Szepesvari, C. (2006). Bandit-based Monte-Carlo planning. In *ECML-06*.

Koditschek, D. (1987). Exact robot navigation by means of potential functions: some topological considerations. In *ICRA-87*, Vol. 1, pp. 1–6.

Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset. In *ECP-97*, pp. 273–285.

Koehn, P. (2009). *Statistical Machine Translation*. Cambridge University Press.

Koenderink, J. J. (1990). *Solid Shape*. MIT Press.

Koenig, S. (1991). Optimal probabilistic and decision-theoretic planning using Markovian decision

theory. Master's report, Computer Science Division, University of California.

Koenig, S. (2000). Exploring unknown environments with real-time search or reinforcement learning. In Solla, S. A., Leen, T. K., and Müller, K.-R. (Eds.), *NIPS 12*. MIT Press.

Koenig, S. (2001). Agent-centered search. *AIMag*, *22*(4), 109–131.

Koller, D., Meggido, N., and von Stengel, B. (1996). Efficient computation of equilibria for extensive two-person games. *Games and Economic Behaviour*, *14*(2), 247–259.

Koller, D. and Pfeffer, A. (1997). Representations and solutions for game-theoretic problems. *AIJ*, *94*(1–2), 167–215.

Koller, D. and Pfeffer, A. (1998). Probabilistic frame-based systems. In *AAAI-98*, pp. 580–587.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Koller, D. and Milch, B. (2003). Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, *45*, 181–221.

Koller, D. and Parr, R. (2000). Policy iteration for factored MDPs. In *UAI-00*, pp. 326–334.

Koller, D. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In *ICML-97*, pp. 170–178.

Kolmogorov, A. N. (1941). Interpolation und extrapolation von stationaren zufalligen folgen. *Bulletin of the Academy of Sciences of the USSR*, *Ser. Math. 5*, 3–14.

Kolmogorov, A. N. (1950). *Foundations of the Theory of Probability*. Chelsea.

Kolmogorov, A. N. (1963). On tables of random numbers. *Sankhya, the Indian Journal of Statistics*, *Series A 25*.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. *Problems in Information Transmission*, *1*(1), 1–7.

Kolodner, J. (1983). Reconstructive memory: A computer model. *Cognitive Science*, *7*, 281–328.

Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann.

Kondrak, G. and van Beek, P. (1997). A theoretical evaluation of selected backtracking algorithms. *AIJ*, *89*, 365–387.

Konolige, K. (1997). COLBERT: A language for reactive control in Saphira. In *Künstliche Intelligenz: Advances in Artificial Intelligence*, LNAI, pp. 31–52.

Konolige, K. (2004). Large-scale map-making. In *AAAI-04*, pp. 457–463.

Konolige, K. (1982). A first order formalization of knowledge and action for a multi-agent planning system. In Hayes, J. E., Michie, D., and Pao, Y.-H. (Eds.), *Machine Intelligence 10*. Ellis Horwood.

Konolige, K. (1994). Easy to be hard: Difficult problems for greedy algorithms. In *KR-94*, pp. 374–378.

Koo, T., Carreras, X., and Collins, M. (2008). Simple semi-supervised dependency parsing. In *ACL-08*.

Koopmans, T. C. (1972). Representation of preference orderings over time. In McGuire, C. B. and Radner, R. (Eds.), *Decision and Organization*. Elsevier/North-Holland.

Korb, K. B. and Nicholson, A. (2003). *Bayesian Artificial Intelligence*. Chapman and Hall.

Korb, K. B., Nicholson, A., and Jitnah, N. (1999). Bayesian poker. In *UAI-99*.

Korf, R. E. (1985a). Depth-first iterative-deepening: an optimal admissible tree search. *AIJ*, *27(1)*, 97–109.

Korf, R. E. (1985b). Iterative-deepening A*: An optimal admissible tree search. In *IJCAI-85*, pp. 1034– 1036.

Korf, R. E. (1987). Planning as search: A quantitative approach. *AIJ*, *33(1)*, 65–88.

Korf, R. E. (1990). Real-time heuristic search. *AIJ*, *42*(3), 189–212.

Korf, R. E. (1993). Linear-space best-first search. *AIJ*, *62*(1), 41–78.

Korf, R. E. (1995). Space-efficient search algorithms. *ACM Computing Surveys*, *27*(3), 337–339.

Korf, R. E. and Chickering, D. M. (1996). Best-first minimax search. *AIJ*, *84*(1–2), 299–337.

Korf, R. E. and Felner, A. (2002). Disjoint pattern database heuristics. *AIJ*, *134*(1–2), 9–22.

Korf, R. E., Reid, M., and Edelkamp, S. (2001). Time complexity of iterative-deepening-A*. *AIJ*, *129*, 199–218.

Korf, R. E. and Zhang, W. (2000). Divide-andconquer frontier search applied to optimal sequence alignment. In *American Association for Artificial Intelligence*, pp. 910–916.

Korf, R. E. (2008). Linear-time disk-based implicit graph search. *JACM*, *55*(6).

Korf, R. E. and Schultze, P. (2005). Large-scale parallel breadth-first search. In *AAAI-05*, pp. 1380–1385.

Kotok, A. (1962). A chess playing program for the IBM 7090. AI project memo 41, MIT Computation Center.

Koutsoupias, E. and Papadimitriou, C. H. (1992). On the greedy algorithm for satisfiability. *Information Processing Letters*, *43*(1), 53–55.

Kowalski, R. (1974). Predicate logic as a programming language. In *Proc. IFIP Congress*, pp. 569–574.

Kowalski, R. (1979). *Logic for Problem Solving*. Elsevier/North-Holland.

Kowalski, R. (1988). The early years of logic programming. *CACM*, *31*, 38–43.

Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, *4*(1), 67–95.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of*

Natural Selection. MIT Press.

Koza, J. R. (1994). *Genetic Programming II: Automatic discovery of reusable programs*. MIT Press.

Koza, J. R., Bennett, F. H., Andre, D., and Keane, M. A. (1999). *Genetic Programming III: Darwinian invention and problem solving*. Morgan Kaufmann.

Kraus, S., Ephrati, E., and Lehmann, D. (1991). Negotiation in a non-cooperative environment. *AIJ*, *3*(4), 255–281.

Krause, A. and Guestrin, C. (2009). Optimal value of information in graphical models. *JAIR*, *35*, 557–591.

Krause, A., McMahan, B., Guestrin, C., and Gupta, A. (2008). Robust submodular observation selection. *JMLR*, *9*, 2761–2801.

Kripke, S. A. (1963). Semantical considerations on modal logic. *Acta Philosophica Fennica*, *16*, 83–94.

Krogh, A., Brown, M., Mian, I. S., Sjolander, K., and Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *J. Molecular Biology*, *235*, 1501–1531.

Kübler, S., McDonald, R., and Nivre, J. (2009). *Dependency Parsing*. Morgan Claypool.

Kuhn, H. W. (1953). Extensive games and the problem of information. In Kuhn, H. W. and Tucker, A. W. (Eds.), *Contributions to the Theory of Games II*. Princeton University Press.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, *2*, 83–97.

Kuipers, B. J. (1985). Qualitative simulation. In Bo-brow, D. (Ed.), *Qualitative Reasoning About Physical Systems*, pp. 169–203. MIT Press.

Kuipers, B. J. and Levitt, T. S. (1988). Navigation and mapping in large-scale space. *AIMag*, *9*(2), 25–43.

Kuipers, B. J. (2001). Qualitative simulation. In Meyers, R. A. (Ed.), *Encyclopeida of Physical Science and Technology*. Academic Press.

Kumar, P. R. and Varaiya, P. (1986). *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice-Hall.

Kumar, V. (1992). Algorithms for constraint satisfaction problems: A survey. *AIMag*, *13*(1), 32–44.

Kumar, V. and Kanal, L. N. (1983). A general branch and bound formulation for understanding and synthesizing and/or tree search procedures. *AIJ*, *21*, 179–198.

Kumar, V. and Kanal, L. N. (1988). The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In Kanal, L. N. and Kumar, V. (Eds.), *Search in Artificial Intelligence*, chap. 1, pp. 1–27. Springer-Verlag.

Kumar, V., Nau, D. S., and Kanal, L. N. (1988). A general branch-and-bound formulation for AND/OR graph and game tree search. In Kanal, L. N. and Kumar, V. (Eds.), *Search in Artificial*

Intelligence, chap. 3, pp. 91–130. Springer-Verlag.

Kurien, J., Nayak, P., and Smith, D. E. (2002). Fragment-based conformant planning. In *AIPS-02*.

Kurzweil, R. (1990). *The Age of Intelligent Machines*. MIT Press.

Kurzweil, R. (2005). *The Singularity is Near*. Viking.

Kwok, C., Etzioni, O., and Weld, D. S. (2001). Scaling question answering to the web. In *Proc. 10th International Conference on the World Wide Web*.

Kyburg, H. E. and Teng, C.-M. (2006). Nonmonotonic logic and statistical inference. *Computational Intelligence*, *22*(1), 26–51.

Kyburg, H. E. (1977). Randomness and the right reference class. *J. Philosophy*, *74*(9), 501–521.

Kyburg, H. E. (1983). The reference class. *Philosophy of Science*, *50*, 374–397.

La Mettrie, J. O. (1748). *L'homme machine*. E. Luzac, Leyde, France.

La Mura, P. and Shoham, Y. (1999). Expected utility networks. In *UAI-99*, pp. 366–373.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling. *AIJ*, *143*(2), 151–188.

Ladkin, P. (1986a). Primitives and units for time specification. In *AAAI-86*, Vol. 1, pp. 354–359.

Ladkin, P. (1986b). Time representation: a taxonomy of interval relations. In *AAAI-86*, Vol. 1, pp. 360–366.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML-01*.

Lafferty, J. and Zhai, C. (2001). Probabilistic relevance models based on document and query generation. In *Proc. Workshop on Language Modeling and Information Retrieval*.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *JMLR*, *4*, 1107–1149.

Laird, J., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *AIJ*, *33*(1), 1–64.

Laird, J., Rosenbloom, P. S., and Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Laird, J. (2008). Extending the Soar cognitive architecture. In *Artificial General Intelligence Conference*.

Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal About the Mind*. University of Chicago Press.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press.

Lakoff, G. and Johnson, M. (1999). *Philosophy in the Flesh : The Embodied Mind and Its Challenge to Western Thought*. Basic Books.

Lam, J. and Greenspan, M. (2008). Eye-in-hand visual servoing for accurate shooting in pool

robotics. In *5th Canadian Conference on Computer and Robot Vision*.

Lamarck, J. B. (1809). *Philosophie zoologique*. Chez Dentu et L'Auteur, Paris.

Landhuis, E. (2004). Lifelong debunker takes on arbiter of neutral choices: Magician-turnedmathematician uncovers bias in a flip of a coin. *Stanford Report*.

Langdon, W. and Poli, R. (2002). *Foundations of Genetic Programming*. Springer.

Langley, P., Simon, H. A., Bradshaw, G. L., and Zytkow, J. M. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes*. MIT Press.

Langton, C. (Ed.). (1995). *Artificial Life*. MIT Press.

Laplace, P. (1816). *Essai philosophique sur les probabilités* (3rd edition). Courcier Imprimeur, Paris.

Laptev, I. and Perez, P. (2007). Retrieving actions in movies. In *ICCV*, pp. 1–8.

Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, *4*, 35–56.

Larrañaga, P., Kuijipers, C., Murga, R., Inza, I., and Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review, 13,* 129-170.

Larson, S. C. (1931). The shrinkage of the coefficient of multiple correlation. *J. Educational Psychology*, *22*, 45–55.

Laskey, K. B. (2008). MEBN: A language for first-order bayesian knowledge bases. *AIJ*, *172*, 140–178.

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer.

Lauritzen, S. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, *19*, 191–201.

Lauritzen, S. (1996). *Graphical models*. Oxford University Press.

Lauritzen, S., Dawid, A. P., Larsen, B., and Leimer, H. (1990). Independence properties of directed Markov fields. *Networks*, *20*(5), 491–505.

Lauritzen, S. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society*, *B 50*(2), 157–224.

Lauritzen, S. and Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, *17*, 31–57.

LaValle, S. (2006). *Planning Algorithms*. Cambridge University Press.

Lavrauc, N. and Duzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

Lawler, E. L., Lenstra, J. K., Kan, A., and Shmoys, D. B. (1992). *The Travelling Salesman Problem*. Wiley Interscience.

Lawler, E. L., Lenstra, J. K., Kan, A., and Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. In Graves, S. C., Zipkin, P. H., and Kan, A. H. G. R. (Eds.), *Logistics of Production and Inventory: Handbooks in Operations Research and Management Science, Volume 4*, pp. 445–522. North-Holland.

Lawler, E. L. and Wood, D. E. (1966). Branch-andbound methods: A survey. *Operations Research*, *14(4)*, 699–719.

Lazanas, A. and Latombe, J.-C. (1992). Landmark-based robot navigation. In *AAAI-92*, pp. 816–822.

LeCun, Y., Jackel, L., Boser, B., and Denker, J. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, *27*(11), 41–46.

LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., Simard, P., and Vapnik, V. N. (1995). Comparison of learning algorithms for handwritten digit recognition. In *Int. Conference on Artificial Neural Networks*, pp. 53–60.

Leech, G., Rayson, P., and Wilson, A. (2001). *Word Frequencies in Written and Spoken English: Based on the British National Corpus*. Longman.

Legendre, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes*..

Lehrer, J. (2009). *How We Decide*. Houghton Mifflin.

Lenat, D. B. (1983). EURISKO: A program that learns new heuristics and domain concepts: The nature of heuristics, III: Program design and results. *AIJ*, *21*(1–2), 61–98.

Lenat, D. B. and Brown, J. S. (1984). Why AM and EURISKO appear to work. *AIJ*, *23*(3), 269–294.

Lenat, D. B. and Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley.

Leonard, H. S. and Goodman, N. (1940). The calculus of individuals and its uses. *JSL*, *5*(2), 45–55.

Leonard, J. and Durrant-Whyte, H. (1992). *Directed sonar sensing for mobile robot navigation*. Kluwer.

Lésniewski, S. (1916). Podstawy og´olnej teorii mnogósci. Moscow.

Lettvin, J. Y., Maturana, H. R., McCulloch, W. S., and Pitts, W. (1959). What the frog's eye tells the frog's brain. *Proc. IRE*, *47*(11), 1940–1951.

Letz, R., Schumann, J., Bayerl, S., and Bibel, W. (1992). SETHEO: A high-performance theorem prover. *JAR*, *8*(2), 183–212.

Levesque, H. J. and Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, *3*(2), 78–93.

Levin, D. A., Peres, Y., and Wilmer, E. L. (2008). *Markov Chains and Mixing Times*. American Mathematical Society.

Levitt, G. M. (2000). *The Turk, Chess Automaton*. McFarland and Company.

Levy, D. (Ed.). (1988a). *Computer Chess Compendium*. Springer-Verlag.

Levy, D. (Ed.). (1988b). *Computer Games*. Springer-Verlag.

Levy, D. (1989). The million pound bridge program. In Levy, D. and Beal, D. (Eds.), *Heuristic Programming in Artificial Intelligence*. Ellis Horwood.

Levy, D. (2007). *Love and Sex with Robots*. Harper.

Lewis, D. D. (1998). Naive Bayes at forty: The independence assumption in information retrieval. In *ECML-98*, pp. 4–15.

Lewis, D. K. (1966). An argument for the identity theory. *J. Philosophy*, *63*(1), 17–25.

Lewis, D. K. (1980). Mad pain and Martian pain. In Block, N. (Ed.), *Readings in Philosophy of Psychology*, Vol. 1, pp. 216–222. Harvard University Press.

Leyton-Brown, K. and Shoham, Y. (2008). *Essentials of Game Theory: A Concise, Multidisciplinary Introduction*. Morgan Claypool.

Li, C. M. and Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *IJCAI-97*, pp. 366–371.

Li, M. and Vitanyi, P. M. B. (1993). *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag.

Liberatore, P. (1997). The complexity of the language **A**. *Electronic Transactions on Artificial Intelligence*, *1*, 13–38.

Lifschitz, V. (2001). Answer set programming and plan generation. *AIJ*, *138*(1–2), 39–54.

Lighthill, J. (1973). Artificial intelligence: A general survey. In Lighthill, J., Sutherland, N. S., Needham, R. M., Longuet-Higgins, H. C., and Michie, D. (Eds.), *Artificial Intelligence: A Paper Symposium*. Science Research Council of Great Britain.

Lin, S. (1965). Computer solutions of the travelling salesman problem. *Bell Systems Technical Journal*, *44(10)*, 2245–2269.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the travelling-salesman problem. *Operations Research*, *21*(2), 498–516.

Lindley, D. V. (1956). On a measure of the information provided by an experiment. *Annals of Mathematical Statistics*, *27*(4), 986–1005.

Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. (1980). *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. McGraw-Hill.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *ICML-94*, pp. 157–163.

Littman, M. L., Keim, G. A., and Shazeer, N. M. (1999). Solving crosswords with PROVERB. In *AAAI-99*, pp. 914–915.

Liu, J. S. and Chen, R. (1998). Sequential Monte Carlo methods for dynamic systems. *JASA*, *93*, 1022–1031.

Livescu, K., Glass, J., and Bilmes, J. (2003). Hidden feature modeling for speech recognition using dynamic Bayesian networks. In *EUROSPEECH-2003*, pp. 2529–2532.

Livnat, A. and Pippenger, N. (2006). An optimal brain can be composed of conflicting agents. *PNAS*, *103*(9), 3198–3202.

Locke, J. (1690). *An Essay Concerning Human Understanding*. William Tegg.

Lodge, D. (1984). *Small World*. Penguin Books.

Loftus, E. and Palmer, J. (1974). Reconstruction of automobile destruction: An example of the interaction between language and memory. *J. Verbal Learning and Verbal Behavior*, *13*, 585–589.

Lohn, J. D., Kraus, W. F., and Colombano, S. P. (2001). Evolutionary optimization of yagi-uda antennas. In *Proc. Fourth International Conference on Evolvable Systems*, pp. 236–243.

Longley, N. and Sankaran, S. (2005). The NHL's overtime-loss rule: Empirically analyzing the unintended effects. *Atlantic Economic Journal*.

Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, *293*, 133–135.

Loo, B. T., Condie, T., Garofalakis, M., Gay, D. E., Hellerstein, J. M., Maniatis, P., Ramakrishnan, R., Roscoe, T., and Stoica, I. (2006). Declarative networking: Language, execution and optimization. In *SIGMOD-06*.

Love, N., Hinrichs, T., and Genesereth, M. R. (2006). General game playing: Game description language specification. Tech. rep. LG-2006-01, Stanford University Computer Science Dept.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, *28*(1–4), 47–66.

Loveland, D. (1970). A linear format for resolution. In *Proc. IRIA Symposium on Automatic Demonstration*, pp. 147–162.

Lowe, D. (1987). Three-dimensional object recognition from single two-dimensional images. *AIJ*, *31*, 355–395.

Lowe, D. (1999). Object recognition using local scale invariant feature. In *ICCV*.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *IJCV*, *60*(2), 91–110.

Löwenheim, L. (1915). Über möglichkeiten im Relativkalkül. *Mathematische Annalen*, *76*, 447–470.

Lowerre, B. T. (1976). *The* HARPY *Speech Recognition System*. Ph.D. thesis, Computer Science Department, Carnegie-Mellon University.

Lowerre, B. T. and Reddy, R. (1980). The HARPY speech recognition system. In Lea, W. A. (Ed.), *Trends in Speech Recognition*, chap. 15. Prentice-Hall.

Lowry, M. (2008). Intelligent software engineering tools for NASA's crew exploration vehicle. In *Proc. ISMIS*.

Loyd, S. (1959). *Mathematical Puzzles of Sam Loyd: Selected and Edited by Martin Gardner*. Dover.

Lozano-Perez, T. (1983). Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, *C-32*(2), 108–120.

Lozano-Perez, T., Mason, M., and Taylor, R. (1984). Automatic synthesis of fine-motion strategies for robots. *Int. J. Robotics Research*, *3*(1), 3–24.

Lu, F. and Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, *4*, 333–349.

Luby, M., Sinclair, A., and Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, *47*, 173–180.

Lucas, J. R. (1961). Minds, machines, and G¨odel. *Philosophy*, *36*.

Lucas, J. R. (1976). This G¨odel is killing me: A rejoinder. *Philosophia*, *6*(1), 145–148.

Lucas, P. (1996). Knowledge acquisition for decision-theoretic expert systems. *AISB Quarterly*, *94*, 23–33.

Lucas, P., van der Gaag, L., and Abu-Hanna, A. (2004). Bayesian networks in biomedicine and health-care. *Artificial Intelligence in Medicine*.

Luce, D. R. and Raiffa, H. (1957). *Games and Decisions*. Wiley.

Ludlow, P., Nagasawa, Y., and Stoljar, D. (2004). *There's Something About Mary*. MIT Press.

Luger, G. F. (Ed.). (1995). *Computation and intelligence: Collected readings*. AAAI Press.

Lyman, P. and Varian, H. R. (2003). How much information? www.sims.berkeley.edu/how-much-info-2003.

Machina, M. (2005). Choice under uncertainty. In *Encyclopedia of Cognitive Science*, pp. 505–514. Wiley.

MacKay, D. J. C. (1992). A practical Bayesian framework for back-propagation networks. *Neural Computation*, *4*(3), 448–472.

MacKay, D. J. C. (2002). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

MacKenzie, D. (2004). *Mechanizing Proof*. MIT Press.

Mackworth, A. K. (1977). Consistency in networks of relations. *AIJ*, *8*(1), 99–118.

Mackworth, A. K. (1992). Constraint satisfaction. In Shapiro, S. (Ed.), *Encyclopedia of Artificial Intelligence* (second edition)., Vol. 1, pp. 285–293. Wiley.

Mahanti, A. and Daniels, C. J. (1993). A SIMD approach to parallel heuristic search. *AIJ*, *60*(2), 243–282.

Mailath, G. and Samuelson, L. (2006). *Repeated Games and Reputations: Long-Run Relationships*. Oxford University Press.

Majercik, S. M. and Littman, M. L. (2003). Contingent planning under uncertainty via stochastic satisfiability. *AIJ*, pp. 119–162.

Malik, J. and Perona, P. (1990). Preattentive texture discrimination with early vision mechanisms. *J. Opt. Soc. Am. A*, *7*(5), 923–932.

Malik, J. and Rosenholtz, R. (1994). Recovering surface curvature and orientation from texture distortion: A least squares algorithm and sensitivity analysis. In *ECCV*, pp. 353–364.

Malik, J. and Rosenholtz, R. (1997). Computing local surface orientation and shape from texture for curved surfaces. *IJCV*, *23*(2), 149–168.

Maneva, E., Mossel, E., and Wainwright, M. J. (2007). A new look at survey propagation and its generalizations. *JACM*, *54*(4).

Manna, Z. and Waldinger, R. (1971). Toward automatic program synthesis. *CACM*, *14*(3), 151–165.

Manna, Z. and Waldinger, R. (1985). *The Logical Basis for Computer Programming: Volume 1: Deductive Reasoning*. Addison-Wesley.

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Manning, C., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Mannion, M. (2002). Using first-order logic for product line model validation. In *Software Product Lines: Second International Conference*. Springer.

Manzini, G. (1995). BIDA*: An improved perimeter search algorithm. *AIJ*, *72*(2), 347–360.

Marbach, P. and Tsitsiklis, J. N. (1998). Simulation-based optimization of Markov reward processes. Technical report LIDS-P-2411, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

Marcus, G. (2009). *Kluge: The Haphazard Evolution of the Human Mind*. Mariner Books.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, *19*(2), 313–330.

Markov, A. A. (1913). An example of statistical investigation in the text of "Eugene Onegin" illustrating coupling of "tests" in chains. *Proc. Academy of Sciences of St. Petersburg*, *7*.

Maron, M. E. (1961). Automatic indexing: An experimental inquiry. *JACM*, *8*(3), 404–417.

Maron, M. E. and Kuhns, J.-L. (1960). On relevance, probabilistic indexing and information retrieval. *CACM*, *7*, 219–244.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman.

Marriott, K. and Stuckey, P. J. (1998). *Programming with Constraints: An Introduction*. MIT Press.

Marsland, A. T. and Schaeffer, J. (Eds.). (1990). *Computers, Chess, and Cognition*. Springer-Verlag.

Marsland, S. (2009). *Machine Learning: An Algorithmic Perspective*. CRC Press.

Martelli, A. and Montanari, U. (1973). Additive AND/OR graphs. In *IJCAI-73*, pp. 1–11.

Martelli, A. and Montanari, U. (1978). Optimizing decision trees through heuristically guided search. *CACM*, *21*, 1025–1039.

Martelli, A. (1977). On the complexity of admissible search algorithms. *AIJ*, *8*(1), 1–13.

Marthi, B., Pasula, H., Russell, S. J., and Peres, Y. (2002). Decayed MCMC filtering. In *UAI-02*, pp. 319–326.

Marthi, B., Russell, S. J., Latham, D., and Guestrin, C. (2005). Concurrent hierarchical reinforcement learning. In *IJCAI-05*.

Marthi, B., Russell, S. J., and Wolfe, J. (2007). Angelic semantics for high-level actions. In *ICAPS-07*.

Marthi, B., Russell, S. J., and Wolfe, J. (2008). Angelic hierarchical planning: Optimal and online algorithms. In *ICAPS-08*.

Martin, D., Fowlkes, C., and Malik, J. (2004). Learning to detect natural image boundaries using local brightness, color, and texture cues. *PAMI*, *26*(5), 530–549.

Martin, J. H. (1990). *A Computational Model of Metaphor Interpretation*. Academic Press.

Mason, M. (1993). Kicking the sensing habit. *AIMag*, *14*(1), 58–59.

Mason, M. (2001). *Mechanics of Robotic Manipulation*. MIT Press.

Mason, M. and Salisbury, J. (1985). *Robot hands and the mechanics of manipulation*. MIT Press.

Mataric, M. J. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, *4*(1), 73–83.

Mates, B. (1953). *Stoic Logic*. University of California Press.

Matuszek, C., Cabral, J., Witbrock, M., and DeOliveira, J. (2006). An introduction to the syntax and semantics of cyc. In *Proc. AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*.

Maxwell, J. and Kaplan, R. (1993). The interface between phrasal and functional constraints. *Computational Linguistics*, *19*(4), 571–590.

McAllester, D. A. (1980). An outlook on truth maintenance. Ai memo 551, MIT AI Laboratory.

McAllester, D. A. (1988). Conspiracy numbers for min-max search. *AIJ*, *35*(3), 287–310.

McAllester, D. A. (1998). What is the most pressing issue facing AI and the AAAI today? Candidate statement, election for Councilor of the American Association for Artificial Intelligence.

McAllester, D. A. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *AAAI-91*, Vol. 2, pp.

634–639.

McCallum, A. (2003). Efficiently inducing features of conditional random fields. In *UAI-03*.

McCarthy, J. (1958). Programs with common sense. In *Proc. Symposium on Mechanisation of Thought Processes*, Vol. 1, pp. 77–84.

McCarthy, J. (1963). Situations, actions, and causal laws. Memo 2, Stanford University Artificial Intelligence Project.

McCarthy, J. (1968). Programs with common sense. In Minsky, M. L. (Ed.), *Semantic Information Processing*, pp. 403–418. MIT Press.

McCarthy, J. (1980). Circumscription: A form of non-monotonic reasoning. *AIJ*, *13*(1–2), 27–39.

McCarthy, J. (2007). From here to human-level AI. *AIJ*, *171*(18), 1174–1182.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., Michie, D., and Swann, M. (Eds.), *Machine Intelligence 4*, pp. 463–502. Edinburgh University Press.

McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C. E. (1955). Proposal for the Dartmouth summer research project on artificial intelligence. Tech. rep., Dartmouth College.

McCawley, J. D. (1988). *The Syntactic Phenomena of English*, Vol. 2 volumes. University of Chicago Press.

McCorduck, P. (2004). *Machines who think: a personal inquiry into the history and prospects of artificial intelligence* (Revised edition). A K Peters.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–137.

McCune, W. (1992). Automated discovery of new axiomatizations of the left group and right group calculi. *JAR*, *9*(1), 1–24.

McCune, W. (1997). Solution of the Robbins problem. *JAR*, *19*(3), 263–276.

McDermott, D. (1976). Artificial intelligence meets natural stupidity. *SIGART Newsletter*, *57*, 4–9.

McDermott, D. (1978a). Planning and acting. *Cognitive Science*, *2*(2), 71–109.

McDermott, D. (1978b). Tarskian semantics, or, no notation without denotation! *Cognitive Science*, *2*(3).

McDermott, D. (1985). Reasoning about plans. In Hobbs, J. and Moore, R. (Eds.), *Formal theories of the commonsense world*. Intellect Books.

McDermott, D. (1987). A critique of pure reason. *Computational Intelligence*, *3*(3), 151–237.

McDermott, D. (1996). A heuristic estimator for means-ends analysis in planning. In *ICAPS-96*, pp. 142–149.

McDermott, D. and Doyle, J. (1980). Non-monotonic logic: i. *AIJ*, *13*(1–2), 41–72.

McDermott, J. (1982). R1: A rule-based configurer of computer systems. *AIJ*, *19*(1), 39–88.

McEliece, R. J., MacKay, D. J. C., and Cheng, J. F. (1998). Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, *16*(2), 140–152.

McGregor, J. J. (1979). Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, *19*(3), 229–250.

McIlraith, S. and Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems*, *16*(2), 46–53.

McLachlan, G. J. and Krishnan, T. (1997). *The EM Algorithm and Extensions*. Wiley.

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer.

Meehl, P. (1955). *Clinical vs. Statistical Prediction*. University of Minnesota Press.

Mendel, G. Versuche uber pflanzen (1866). ¨hybriden. *Verhandlungen des Naturforschenden Vereins, Abhandlungen, Brünn*, *4*, 3–47. Translated into English by C. T. Druery, published by Bateson (1902).

Mercer, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London, A*, *209*, 415–446.

Merleau-Ponty, M. (1945). *Phenomenology of Perception*. Routledge.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equations of state calculations by fast computing machines. *J. Chemical Physics*, *21*, 1087–1091.

Metzinger, T. (2009). *The Ego Tunnel: The Science of the Mind and the Myth of the Self*. Basic Books.

Mézard, M. and Nadal, J.-P. (1989). Learning in feedforward layered networks: The tiling algorithm. *J. Physics*, *22*, 2191–2204.

Michalski, R. S. (1969). On the quasi-minimal solution of the general covering problem. In *Proc. First International Symposium on Information Processing*, pp. 125–128.

Michalski, R. S., Mozetic, I., Hong, J., and Lavra**uc**, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *AAAI-86*, pp. 1041–1045.

Michie, D. (1966). Game-playing and game-learning automata. In Fox, L. (Ed.), *Advances in Programming and Non-Numerical Computation*, pp. 183–200. Pergamon.

Michie, D. (1972). Machine intelligence at Edinburgh. *Management Informatics*, *2*(1), 7–12.

Michie, D. (1974). Machine intelligence at Edinburgh. In *On Intelligence*, pp. 143–155. Edinburgh University Press.

Michie, D. and Chambers, R. A. (1968). BOXES: An experiment in adaptive control. In Dale, E. and Michie, D. (Eds.), *Machine Intelligence 2*, pp. 125–133. Elsevier/North-Holland.

Michie, D., Spiegelhalter, D. J., and Taylor, C. (Eds.). (1994). *Machine Learning, Neural and*

Statistical Classification. Ellis Horwood.

Milch, B., Marthi, B., Sontag, D., Russell, S. J., Ong, D., and Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In *IJCAI-05*.

Milch, B., Zettlemoyer, L. S., Kersting, K., Haimes, M., and Kaelbling, L. P. (2008). Lifted probabilistic inference with counting formulas. In *AAAI-08*, pp. 1062–1068.

Milgrom, P. (1997). Putting auction theory to work: The simultaneous ascending auction. Tech. rep. Technical Report 98-0002, Stanford University Department of Economics.

Mill, J. S. (1843). *A System of Logic, Ratiocinative and Inductive: Being a Connected View of the Principles of Evidence, and Methods of Scientific Investigation*. J. W. Parker, London.

Mill, J. S. (1863). *Utilitarianism*. Parker, Son and Bourn, London.

Miller, A. C., Merkhofer, M. M., Howard, R. A., Matheson, J. E., and Rice, T. R. (1976). Development of automated aids for decision analysis. Technical report, SRI International.

Minker, J. (2001). *Logic-Based Artificial Intelligence*. Kluwer.

Minsky, M. L. (1975). A framework for representing knowledge. In Winston, P. H. (Ed.), *The Psychology of Computer Vision*, pp. 211–277. McGraw-Hill. Originally an MIT AI Laboratory memo; the 1975 version is abridged, but is the most widely cited.

Minsky, M. L. (1986). *The society of mind*. Simon and Schuster.

Minsky, M. L. (2007). *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. Simon and Schuster.

Minsky, M. L. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry* (first edition). MIT Press.

Minsky, M. L. and Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry* (Expanded edition). MIT Press.

Minsky, M. L., Singh, P., and Sloman, A. (2004). The st. thomas common sense symposium: Designing architectures for human-level intelligence. *AIMag*, *25*(2), 113–124.

Minton, S. (1984). Constraint-based generalization: Learning game-playing plans from single examples. In *AAAI-84*, pp. 251–254.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *AAAI88*, pp. 564–569.

Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *AIJ*, *58*(1–3), 161–205.

Misak, C. (2004). *The Cambridge Companion to Peirce*. Cambridge University Press.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Mitchell, M., Holland, J. H., and Forrest, S. (1996). When will a genetic algorithm outperform hill climbing? In Cowan, J., Tesauro, G., and Alspector, J. (Eds.), *NIPS 6*. MIT Press.

Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule learning. In *IJCAI-77*, pp. 305–310.

Mitchell, T. M. (1982). Generalization as search. *AIJ*, *18*(2), 203–226.

Mitchell, T. M. (1990). Becoming increasingly reactive (mobile robots). In *AAAI-90*, Vol. 2, pp. 1051– 1058.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Mitchell, T. M., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, *1*, 47–80.

Mitchell, T. M., Utgoff, P. E., and Banerji, R. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, pp. 163–190. Morgan Kaufmann.

Mitchell, T. M. (2005). Reading the web: A breakthrough goal for AI. *AIMag*, *26*(3), 12–16.

Mitchell, T. M. (2007). Learning, information extraction and the web. In *ECML/PKDD*, p. 1.

Mitchell, T. M., Shinkareva, S. V., Carlson, A., Chang, K.-M., Malave, V. L., Mason, R. A., and Just, M. A. (2008). Predicting human brain activity associated with the meanings of nouns. *Science*, *320*, 1191–1195.

Mohr, R. and Henderson, T. C. (1986). Arc and path consistency revisited. *AIJ*, *28*(2), 225–233.

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, *16*(1), 69–88.

Montague, P. R., Dayan, P., Person, C., and Sejnowski, T. (1995). Bee foraging in uncertain environments using predictive Hebbian learning. *Nature*, *377*, 725–728.

Montague, R. (1970). English as a formal language. In *Linguaggi nella Societ`a e nella Tecnica*, pp. 189–224. Edizioni di Comunit`a.

Montague, R. (1973). The proper treatment of quantification in ordinary English. In Hintikka, K. J. J., Moravcsik, J. M. E., and Suppes, P. (Eds.), *Approaches to Natural Language*. D. Reidel.

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, *7*(2), 95–132.

Montemerlo, M. and Thrun, S. (2004). Large-scale robotic 3-D mapping of urban structures. In *Proc. International Symposium on Experimental Robotics*. Springer Tracts in Advanced Robotics (STAR).

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI-02*.

Mooney, R. (1999). Learning for semantic interpretation: Scaling up without dumbing down. In *Proc. 1st Workshop on Learning Language in Logic*, pp. 7–15.

Moore, A. and Wong, W.-K. (2003). Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. In *ICML-03*.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping—Reinforcement learning with less data and less time. *Machine Learning*, *13*, 103–130.

Moore, A. W. and Lee, M. S. (1997). Cached sufficient statistics for efficient machine learning with large datasets. *JAIR*, *8*, 67–91.

Moore, E. F. (1959). The shortest path through a maze. In *Proc. an International Symposium on the Theory of Switching, Part II*, pp. 285–292. Harvard University Press.

Moore, R. C. (1980). Reasoning about knowledge and action. Artificial intelligence center technical note 191, SRI International.

Moore, R. C. (1985). A formal theory of knowledge and action. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, pp. 319–358. Ablex.

Moore, R. C. (2005). Association-based bilingual word alignment. In *Proc. ACL-05 Workshop on Building and Using Parallel Texts*, pp. 1–8.

Moravec, H. P. (1983). The stanford cart and the cmu rover. *Proc. IEEE*, *71*(7), 872–884.

Moravec, H. P. and Elfes, A. (1985). High resolution maps from wide angle sonar. In *ICRA-85*, pp. 116–121.

Moravec, H. P. (1988). *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press.

Moravec, H. P. (2000). *Robot: Mere Machine to Transcendent Mind*. Oxford University Press.

Morgenstern, L. (1998). Inheritance comes of age: Applying nonmonotonic techniques to problems in industry. *AIJ*, *103*, 237–271.

Morjaria, M. A., Rink, F. J., Smith, W. D., Klempner, G., Burns, C., and Stein, J. (1995). Elicitation of probabilities for belief networks: Combining qualitative and quantitative information. In *UAI-95*, pp. 141–148.

Morrison, P. and Morrison, E. (Eds.). (1961). *Charles Babbage and His Calculating Engines: Selected Writings by Charles Babbage and Others*. Dover.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proc. 38th Design Automation Conference (DAC 2001)*, pp. 530–535.

Mosteller, F. and Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist*. Addison-Wesley.

Mostow, J. and Prieditis, A. E. (1989). Discovering admissible heuristics by abstracting and optimizing: A transformational approach. In *IJCAI-89*, Vol. 1, pp. 701–707.

Motzkin, T. S. and Schoenberg, I. J. (1954). The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, *6*(3), 393–404.

Moutarlier, P. and Chatila, R. (1989). Stochastic multisensory data fusion for mobile robot location and environment modeling. In *ISRR-89*.

Mueller, E. T. (2006). *Commonsense Reasoning*. Morgan Kaufmann.

Muggleton, S. H. (1991). Inductive logic programming. *New Generation Computing*, *8*, 295–318.

Muggleton, S. H. (1992). *Inductive Logic Programming*. Academic Press.

Muggleton, S. H. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*(3-4), 245–286.

Muggleton, S. H. (2000). Learning stochastic logic programs. Proc. AAAI 2000 Workshop on Learning Statistical Models from Relational Data.

Muggleton, S. H. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *ICML-88*, pp. 339–352.

Muggleton, S. H. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *J. Logic Programming*, *19/20*, 629–679.

Muggleton, S. H. and Feng, C. (1990). Efficient induction of logic programs. In *Proc. Workshop on Algorithmic Learning Theory*, pp. 368–381.

Müller, M. (2002). Computer Go. *AIJ*, *134*(1–2), 145–179.

Müller, M. (2003). Conditional combinatorial games, and their application to analyzing capturing races in go. *Information Sciences*, *154*(3–4), 189–202.

Mumford, D. and Shah, J. (1989). Optimal approximations by piece-wise smooth functions and associated variational problems. *Commun. Pure Appl. Math.*, *42*, 577–685.

Murphy, K., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *UAI-99*, pp. 467–475.

Murphy, K. (2001). The Bayes net toolbox for MATLAB. *Computing Science and Statistics*, *33*.

Murphy, K. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning*. Ph.D. thesis, UC Berkeley.

Murphy, K. and Mian, I. S. (1999). Modelling gene expression data using Bayesian networks. people.cs.ubc.ca/~murphyk/Papers/ ismb99.pdf.

Murphy, K. and Russell, S. J. (2001). Raoblackwellised particle filtering for dynamic Bayesian networks. In Doucet, A., de Freitas, N., and Gordon, N. J. (Eds.), *Sequential Monte Carlo Methods in Practice*. Springer-Verlag.

Murphy, K. and Weiss, Y. (2001). The factored frontier algorithm for approximate inference in DBNs. In *UAI-01*, pp. 378–385.

Murphy, R. (2000). *Introduction to AI Robotics*. MIT Press.

Murray-Rust, P., Rzepa, H. S., Williamson, J., and Willighagen, E. L. (2003). Chemical markup, XML and the world–wide web. 4. CML schema. *J. Chem. Inf. Comput. Sci.*, *43*, 752–772.

Murthy, C. and Russell, J. R. (1990). A constructive proof of Higman's lemma. In *LICS-90*, pp. 257–269.

Muscettola, N. (2002). Computing the envelope for stepwise-constant resource allocations. In *CP-*

02, pp. 139–154.

Muscettola, N., Nayak, P., Pell, B., and Williams, B. (1998). Remote agent: To boldly go where no AI system has gone before. *AIJ*, *103*, 5–48.

Muslea, I. (1999). Extraction patterns for information extraction tasks: A survey. In *Proc. AAAI-99 Workshop on Machine Learning for Information Extraction*.

Myerson, R. (1981). Optimal auction design. *Mathematics of Operations Research*, *6*, 58–73.

Myerson, R. (1986). Multistage games with communication. *Econometrica*, *54*, 323–358.

Myerson, R. (1991). *Game Theory: Analysis of Conflict*. Harvard University Press.

Nagel, T. (1974). What is it like to be a bat? *Philosophical Review*, *83*, 435–450.

Nalwa, V. S. (1993). *A Guided Tour of Computer Vision*. Addison-Wesley.

Nash, J. (1950). Equilibrium points in N-person games. *PNAS*, *36*, 48–49.

Nau, D. S. (1980). Pathology on game trees: A summary of results. In *AAAI-80*, pp. 102–104.

Nau, D. S. (1983). Pathology on game trees revisited, and an alternative to minimaxing. *AIJ*, *21*(1–2), 221–244.

Nau, D. S., Kumar, V., and Kanal, L. N. (1984). General branch and bound, and its relation to A* and AO*. *AIJ*, *23*, 29–58.

Nayak, P. and Williams, B. (1997). Fast context switching in real-time propositional reasoning. In *AAAI-97*, pp. 50–56.

Neal, R. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag.

Nebel, B. (2000). On the compilability and expressive power of propositional planning formalisms. *JAIR*, *12*, 271–315.

Nefian, A., Liang, L., Pi, X., Liu, X., and Murphy, K. (2002). Dynamic bayesian networks for audiovisual speech recognition. *EURASIP, Journal of Applied Signal Processing*, *11*, 1–15.

Nesterov, Y. and Nemirovski, A. (1994). *Interior-Point Polynomial Methods in Convex Programming*. SIAM (Society for Industrial and Applied Mathematics).

Netto, E. (1901). *Lehrbuch der Combinatorik*. B. G. Teubner.

Nevill-Manning, C. G. and Witten, I. H. (1997). Identifying hierarchical structures in sequences: A linear-time algorithm. *JAIR*, *7*, 67–82.

Newell, A. (1982). The knowledge level. *AIJ*, *18*(1), 82–127.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press.

Newell, A. and Ernst, G. (1965). The search for generality. In *Proc. IFIP Congress*, Vol. 1, pp. 17–24.

Newell, A., Shaw, J. C., and Simon, H. A. (1957). Empirical explorations with the logic theory machine. *Proc. Western Joint Computer Conference*, *15*, 218–239. Reprinted in Feigenbaum and Feldman (1963).

Newell, A., Shaw, J. C., and Simon, H. A. (1958). Chess playing programs and the problem of complexity. *IBM Journal of Research and Development*, *4*(2), 320–335.

Newell, A. and Simon, H. A. (1961). GPS, a program that simulates human thought. In Billing, H. (Ed.), *Lernende Automaten*, pp. 109–124. R. Oldenbourg.

Newell, A. and Simon, H. A. (1972). *Human Problem Solving*. Prentice-Hall.

Newell, A. and Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *CACM*, *19*, 113–126.

Newton, I. (1664–1671). Methodus fluxionum et serierum infinitarum. Unpublished notes.

Ng, A. Y. (2004). Feature selection, l1 vs. l2 regularization, and rotational invariance. In *ICML-04*.

Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML-99*.

Ng, A. Y. and Jordan, M. I. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *UAI-00*, pp. 406–415.

Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2004). Autonomous helicopter flight via reinforcement learning. In *NIPS 16*.

Nguyen, X. and Kambhampati, S. (2001). Reviving partial order planning. In *IJCAI-01*, pp. 459–466.

Nguyen, X., Kambhampati, S., and Nigenda, R. S. (2001). Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. Tech. rep., Computer Science and Engineering Department, Arizona State University.

Nicholson, A. and Brady, J. M. (1992). The data association problem when monitoring robot vehicles using dynamic belief networks. In *ECAI-92*, pp. 689–693.

Niemelä, I., Simons, P., and Syrjänen, T. (2000). Smodels: A system for answer set programming. In *Proc. 8th International Workshop on Non-Monotonic Reasoning*.

Nigam, K., McCallum, A., Thrun, S., and Mitchell, T. M. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning*, *39*(2–3), 103–134.

Niles, I. and Pease, A. (2001). Towards a standard upper ontology. In *FOIS '01: Proc. international conference on Formal Ontology in Information Systems*, pp. 2–9.

Nilsson, D. and Lauritzen, S. (2000). Evaluating influence diagrams using LIMIDs. In *UAI-00*, pp. 436–445.

Nilsson, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill. Republished in 1990.

Nilsson, N. J. (1971). *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.

Nilsson, N. J. (1984). Shakey the robot. Technical note 323, SRI International.

Nilsson, N. J. (1986). Probabilistic logic. *AIJ*, *28*(1), 71–87.

Nilsson, N. J. (1991). Logic and artificial intelligence. *AIJ*, *47*(1–3), 31–56.

Nilsson, N. J. (1995). Eye on the prize. *AIMag*, *16*(2), 9–17.

Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann.

Nilsson, N. J. (2005). Human-level artificial intelligence? be serious! *AIMag*, *26*(4), 68–75.

Nilsson, N. J. (2009). *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press.

Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. (Eds.). (2007). *Algorithmic Game Theory*.

Cambridge University Press.

Noe, A. (2009). *Out of Our Heads: Why You Are Not Your Brain, and Other Lessons from the Biology of Consciousness*. Hill and Wang.

Norvig, P. (1988). Multiple simultaneous interpretations of ambiguous sentences. In *COGSCI-88*.

Norvig, P. (1992). *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann.

Norvig, P. (2009). Natural language corpus data. In Segaran, T. and Hammerbacher, J. (Eds.), *Beautiful Data*. O'Reilly.

Nowick, S. M., Dean, M. E., Dill, D. L., and Horowitz, M. (1993). The design of a high-performance cache controller: A case study in asynchronous synthesis. *Integration: The VLSI Journal*, *15*(3), 241–262.

Nunberg, G. (1979). The non-uniqueness of semantic solutions: Polysemy. *Language and Philosophy*, *3*(2), 143–184.

Nussbaum, M. C. (1978). *Aristotle's* De Motu Animalium. Princeton University Press.

Oaksford, M. and Chater, N. (Eds.). (1998). *Rational models of cognition*. Oxford University Press.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment model. *Computational Linguistics*, *29*(1), 19–51.

Och, F. J. and Ney, H. (2004). The alignment template approach to statistical machine translation. *Computational Linguistics*, *30*, 417–449.

Ogawa, S., Lee, T.-M., Kay, A. R., and Tank, D. W. (1990). Brain magnetic resonance imaging with contrast dependent on blood oxygenation. *PNAS*, *87*, 9868–9872.

Oh, S., Russell, S. J., and Sastry, S. (2009). Markov chain Monte Carlo data association for multi-target tracking. *IEEE Transactions on Automatic Control*, *54*(3), 481–497.

Olesen, K. G. (1993). Causal probabilistic networks with both discrete and continuous variables. *PAMI*, *15*(3), 275–279.

Oliver, N., Garg, A., and Horvitz, E. J. (2004). Layered representations for learning and inferring office activity from multiple sensory channels. *Computer Vision and Image Understanding*, *96*, 163–180.

Oliver, R. M. and Smith, J. Q. (Eds.). (1990). *Influence Diagrams, Belief Nets and Decision Analysis*. Wiley.

Omohundro, S. (2008). The basic AI drives. In *AGI-08 Workshop on the Sociocultural, Ethical and Futurological Implications of Artificial Intelligence*.

O'Reilly, U.-M. and Oppacher, F. (1994). Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In *Proc. Third Conference on Parallel Problem Solving from Nature*, pp. 397–406.

Ormoneit, D. and Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning*, *49*(2–3),

161–178.

Osborne, M. J. (2004). *An Introduction to Game Theory*. Oxford University Pres.

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*. MIT Press.

Osherson, D. N., Stob, M., and Weinstein, S. (1986). *Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press.

Padgham, L. and Winikoff, M. (2004). *Developing Intelligent Agent Systems: A Practical Guide*. Wiley.

Page, C. D. and Srinivasan, A. (2002). ILP: A short look back and a longer look forward. Submitted to Journal of Machine Learning Research.

Palacios, H. and Geffner, H. (2007). From conformant into classical planning: Efficient translations that may be complete too. In *ICAPS-07*.

Palay, A. J. (1985). *Searching with Probabilities*. Pitman.

Palmer, D. A. and Hearst, M. A. (1994). Adaptive sentence boundary disambiguation. In *Proc. Conference on Applied Natural Language Processing*, pp. 78–83.

Palmer, S. (1999). *Vision Science: Photons to Phenomenology*. MIT Press.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison Wesley.

Papadimitriou, C. H., Tamaki, H., Raghavan, P., and Vempala, S. (1998). Latent semantic indexing: A probabilistic analysis. In *PODS-98*, pp. 159–168.

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450.

Papadimitriou, C. H. and Yannakakis, M. (1991). Shortest paths without a map. *Theoretical Computer Science*, *84*(1), 127–150.

Papavassiliou, V. and Russell, S. J. (1999). Convergence of reinforcement learning with general function approximators. In *IJCAI-99*, pp. 748–757.

Parekh, R. and Honavar, V. (2001). DFA learning from simple examples. *Machine Learning*, *44*, 9–35.

Parisi, G. (1988). *Statistical field theory*. Addison-Wesley.

Parisi, M. M. G. and Zecchina, R. (2002). Analytic and algorithmic solution of random satisfiability problems. *Science*, *297*, 812–815.

Parker, A., Nau, D. S., and Subrahmanian, V. S. (2005). Game-tree search with combinatorially large belief states. In *IJCAI-05*, pp. 254–259.

Parker, D. B. (1985). Learning logic. Technical report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology.

Parker, L. E. (1996). On the design of behavior-based multi-robot teams. *J. Advanced Robotics*, *10*(6).

Parr, R. and Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In Jordan, M. I., Kearns, M., and Solla, S. A. (Eds.), *NIPS 10*. MIT Press.

Parzen, E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, *33*, 1065–1076.

Pasca, M. and Harabagiu, S. M. (2001). High performance question/answering. In *SIGIR-01*, pp. 366–374.

Pasca, M., Lin, D., Bigham, J., Lifchits, A., and Jain, A. (2006). Organizing and searching the world wide web of facts—Step one: The one-million fact extraction challenge. In *AAAI-06*.

Paskin, M. (2001). Grammatical bigrams. In *NIPS*.

Pasula, H., Marthi, B., Milch, B., Russell, S. J., and Shpitser, I. (2003). Identity uncertainty and citation matching. In *NIPS 15*. MIT Press.

Pasula, H. and Russell, S. J. (2001). Approximate inference for first-order probabilistic languages. In *IJCAI-01*.

Pasula, H., Russell, S. J., Ostland, M., and Ritov, Y. (1999). Tracking many objects with many sensors. In *IJCAI-99*.

Patashnik, O. (1980). Qubic: 4x4x4 tic-tac-toe. *Mathematics Magazine*, *53*(4), 202–216.

Patrick, B. G., Almulla, M., and Newborn, M. (1992). An upper bound on the time complexity of iterative-deepening-A*. *AIJ*, *5*(2–4), 265–278.

Paul, R. P. (1981). *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press.

Pauls, A. and Klein, D. (2009). K-best A* parsing. In *ACL-09*.

Peano, G. (1889). *Arithmetices principia, nova methodo exposita*. Fratres Bocca, Turin.

Pearce, J., Tambe, M., and Maheswaran, R. (2008). Solving multiagent networks using distributed constraint optimization. *AIMag*, *29*(3), 47–62.

Pearl, J. (1982a). Reverend Bayes on inference engines: A distributed hierarchical approach. In *AAAI82*, pp. 133–136.

Pearl, J. (1982b). The solution for the branching factor of the alpha–beta pruning algorithm and its optimality. *CACM*, *25*(8), 559–564.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *AIJ*, *29*, 241–288.

Pearl, J. (1987). Evidential reasoning using stochastic simulation of causal models. *AIJ*, *32*, 247–257.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Pearl, J. and Verma, T. (1991). A theory of inferred causation. In *KR-91*, pp. 441–452.

Pearson, J. and Jeavons, P. (1997). A survey of tractable constraint satisfaction problems. Technical report CSD-TR-97-15, Royal Holloway College, U. of London.

Pease, A. and Niles, I. (2002). IEEE standard upper ontology: A progress report. *Knowledge Engineering Review*, *17*(1), 65–70.

Pednault, E. P. D. (1986). Formulating multiagent, dynamic-world problems in the classical planning framework. In *Reasoning about Actions and Plans: Proc. 1986 Workshop*, pp. 47–82.

Peirce, C. S. (1870). Description of a notation for the logic of relatives, resulting from an amplification of the conceptions of Boole's calculus of logic. *Memoirs of the American Academy of Arts and Sciences*, *9*, 317–378.

Peirce, C. S. (1883). A theory of probable inference. Note B. The logic of relatives. In *Studies in Logic by Members of the Johns Hopkins University*, pp. 187– 203, Boston.

Peirce, C. S. (1902). Logic as semiotic: The theory of signs. Unpublished manuscript; reprinted in (Buchler 1955).

Peirce, C. S. (1909). Existential graphs. Unpublished manuscript; reprinted in (Buchler 1955).

Pelikan, M., Goldberg, D. E., and Cantu-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In *GECCO-99: Proc. Genetic and Evolutionary Computation Conference*, pp. 525–532.

Pemberton, J. C. and Korf, R. E. (1992). Incremental planning on graphs with cycles. In *AIPS-92*, pp. 525–532.

Penberthy, J. S. and Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *KR-92*, pp. 103–114.

Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, *2*, 437–454.

Penrose, R. (1989). *The Emperor's New Mind*. Oxford University Press.

Penrose, R. (1994). *Shadows of the Mind*. Oxford University Press.

Peot, M. and Smith, D. E. (1992). Conditional nonlinear planning. In *ICAPS-92*, pp. 189–197.

Pereira, F. and Shieber, S. (1987). *Prolog and Natural-Language Analysis*. Center for the Study of Language and Information (CSLI).

Pereira, F. and Warren, D. H. D. (1980). Definite clause grammars for language analysis: A survey of the formalism and a comparison with augmented transition networks. *AIJ*, *13*, 231–278.

Pereira, F. and Wright, R. N. (1991). Finite-state approximation of phrase structure grammars. In *ACL91*, pp. 246–255.

Perlis, A. (1982). Epigrams in programming. *SIGPLAN Notices*, *17*(9), 7–13.

Perrin, B. E., Ralaivola, L., and Mazurie, A. (2003). Gene networks inference using dynamic Bayesian networks. *Bioinformatics*, *19*, II 138–II 148.

Peterson, C. and Anderson, J. R. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems*, *1*(5), 995–1019.

Petrik, M. and Zilberstein, S. (2009). Bilinear programming approach for multiagent planning. *JAIR*, *35*, 235–274.

Petrov, S. and Klein, D. (2007a). Discriminative log-linear grammars with latent variables. In *NIPS*.

Petrov, S. and Klein, D. (2007b). Improved inference for unlexicalized parsing. In *ACL-07*.

Petrov, S. and Klein, D. (2007c). Learning and inference for hierarchically split pcfgs. In *AAAI-07*.

Pfeffer, A., Koller, D., Milch, B., and Takusagawa, K. T. (1999). SPOOK: A system for probabilistic object-oriented knowledge representation. In *UAI 99*.

Pfeffer, A. (2000). *Probabilistic Reasoning for Complex Systems*. Ph.D. thesis, Stanford University.

Pfeffer, A. (2007). The design and implementation of IBAL: A general-purpose probabilistic language. In Getoor, L. and Taskar, B. (Eds.), *Introduction to Statistical Relational Learning*. MIT Press.

Pfeifer, R., Bongard, J., Brooks, R. A., and Iwasawa, S. (2006). *How the Body Shapes the Way We Think: A New View of Intelligence*. Bradford.

Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI-03*.

Pinedo, M. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer Verlag.

Pinkas, G. and Dechter, R. (1995). Improving connectionist energy minimization. *JAIR*, *3*, 223–248.

Pinker, S. (1995). Language acquisition. In Gleitman, L. R., Liberman, M., and Osherson, D. N. (Eds.), *An Invitation to Cognitive Science* (second edition)., Vol. 1. MIT Press.

Pinker, S. (2003). *The Blank Slate: The Modern Denial of Human Nature*. Penguin.

Pinto, D., McCallum, A., Wei, X., and Croft, W. B. (2003). Table extraction using conditional random fields. In *SIGIR-03*.

Pipatsrisawat, K. and Darwiche, A. (2007). RSat 2.0: SAT solver description. Tech. rep. D–153, Automated Reasoning Group, Computer Science Department, University of California, Los Angeles.

Plaat, A., Schaeffer, J., Pijls, W., and de Bruin, A. (1996). Best-first fixed-depth minimax algorithms. *AIJ*, *87*(1–2), 255–293.

Place, U. T. (1956). Is consciousness a brain process? *British Journal of Psychology*, *47*, 44–50.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods: Support Vector Learning*, pp. 185–208. MIT Press.

Plotkin, G. (1971). *Automatic Methods of Inductive Inference*. Ph.D. thesis, Edinburgh University.

Plotkin, G. (1972). Building-in equational theories. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 7*, pp. 73–90. Edinburgh University Press.

Pohl, I. (1971). Bi-directional search. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 6*, pp. 127–140. Edinburgh University Press.

Pohl, I. (1973). The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *IJCAI-73*, pp. 20–23.

Pohl, I. (1977). Practical and theoretical considerations in heuristic search algorithms. In Elcock, E. W. and Michie, D. (Eds.), *Machine Intelligence 8*, pp. 55–72. Ellis Horwood.

Poli, R., Langdon, W., and McPhee, N. (2008). *A Field Guide to Genetic Programming*. Lulu.com.

Pomerleau, D. A. (1993). *Neural Network Perception for Mobile Robot Guidance*. Kluwer.

Ponte, J. and Croft, W. B. (1998). A language modeling approach to information retrieval. In *SIGIR-98*, pp. 275–281.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *AIJ*, *64*, 81–129.

Poole, D. (2003). First-order probabilistic inference. In *IJCAI-03*, pp. 985–991.

Poole, D., Mackworth, A. K., and Goebel, R. (1998). *Computational intelligence: A logical approach*. Oxford University Press.

Popper, K. R. (1959). *The Logic of Scientific Discovery*. Basic Books.

Popper, K. R. (1962). *Conjectures and Refutations: The Growth of Scientific Knowledge*. Basic Books.

Portner, P. and Partee, B. H. (2002). *Formal Semantics: The Essential Readings*. Wiley-Blackwell.

Post, E. L. (1921). Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, *43*, 163–185.

Poundstone, W. (1993). *Prisoner's Dilemma*. Anchor.

Pourret, O., Näım, P., and Marcot, B. (2008). *Bayesian Networks: A practical guide to applications*. Wiley.

Prades, J. L. P., Loomes, G., and Brey, R. (2008). Trying to estmate a monetary value for the QALY. Tech. rep. WP Econ 08.09, Univ. Pablo Olavide.

Pradhan, M., Provan, G. M., Middleton, B., and Henrion, M. (1994). Knowledge engineering for large belief networks. In *UAI-94*, pp. 484–490.

Prawitz, D. (1960). An improved proof procedure. *Theoria*, *26*, 102–139.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing* (third edition). Cambridge University Press.

Preston, J. and Bishop, M. (2002). *Views into the Chinese Room: New Essays on Searle and Artificial Intelligence*. Oxford University Press.

Prieditis, A. E. (1993). Machine discovery of effective admissible heuristics. *Machine Learning*, *12*(1– 3), 117–141.

Prinz, D. G. (1952). Robot chess. *Research*, *5*, 261–266.

Prosser, P. (1993). Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence*, *9*, 268–299.

Pullum, G. K. (1991). *The Great Eskimo Vocabulary Hoax (and Other Irreverent Essays on the Study of Language)*. University of Chicago Press.

Pullum, G. K. (1996). Learnability, hyperlearning, and the poverty of the stimulus. In *22nd Annual Meeting of the Berkeley Linguistics Society*.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Puterman, M. L. and Shin, M. C. (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, *24*(11), 1127–1137.

Putnam, H. (1960). Minds and machines. In Hook, S. (Ed.), *Dimensions of Mind*, pp. 138–164. Macmillan.

Putnam, H. (1963). 'Degree of confirmation' and inductive logic. In Schilpp, P. A. (Ed.), *The Philosophy of Rudolf Carnap*, pp. 270–292. Open Court.

Putnam, H. (1967). The nature of mental states. In Capitan, W. H. and Merrill, D. D. (Eds.), *Art, Mind, and Religion*, pp. 37–48. University of Pittsburgh Press.

Putnam, H. (1975). The meaning of "meaning". In Gunderson, K. (Ed.), *Language, Mind and Knowledge: Minnesota Studies in the Philosophy of Science*. University of Minnesota Press.

Pylyshyn, Z. W. (1974). Minds, machines and phenomenology: Some reflections on Dreyfus' "What Computers Can't Do". *Int. J. Cognitive Psychology*, *3*(1), 57–77.

Pylyshyn, Z. W. (1984). *Computation and Cognition: Toward a Foundation for Cognitive Science*. MIT Press.

Quillian, M. R. (1961). A design for an understanding machine. Paper presented at a colloquium: Semantic Problems in Natural Language, King's College, Cambridge, England.

Quine, W. V. (1953). Two dogmas of empiricism. In *From a Logical Point of View*, pp. 20–46. Harper and Row.

Quine, W. V. (1960). *Word and Object*. MIT Press.

Quine, W. V. (1982). *Methods of Logic* (fourth edition). Harvard University Press.

Quinlan, J. R. (1979). Discovering rules from large collections of examples: A case study. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age*. Edinburgh University Press.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann.

Quinlan, J. R. and Cameron-Jones, R. M. (1993). FOIL: A midterm report. In *ECML-93*, pp. 3–20.

Quirk, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*. Longman.

Rabani, Y., Rabinovich, Y., and Sinclair, A. (1998). A computational view of population genetics. *Random Structures and Algorithms*, *12*(4), 313–334.

Rabiner, L. R. and Juang, B.-H. (1993). *Fundamentals of Speech Recognition*. Prentice-Hall.

Ralphs, T. K., Ladanyi, L., and Saltzman, M. J. (2004). A library hierarchy for implementing scalable parallel search algorithms. *J. Supercomputing*, *28*(2), 215–234.

Ramanan, D., Forsyth, D., and Zisserman, A. (2007). Tracking people by learning their appearance. *IEEE Pattern Analysis and Machine Intelligence*.

Ramsey, F. P. (1931). Truth and probability. In Braithwaite, R. B. (Ed.), *The Foundations of Mathematics and Other Logical Essays*. Harcourt Brace Jovanovich.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In *NIPS 19*, pp. 1137–1144.

Raphson, J. (1690). *Analysis aequationum universalis*. Apud Abelem Swalle, London.

Rashevsky, N. (1936). Physico-mathematical aspects of excitation and conduction in nerves. In *Cold Springs Harbor Symposia on Quantitative Biology. IV: Excitation Phenomena*, pp. 90–97.

Rashevsky, N. (1938). *Mathematical Biophysics: Physico-Mathematical Foundations of Biology*. University of Chicago Press.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Rassenti, S., Smith, V., and Bulfin, R. (1982). A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics*, *13*, 402–417.

Ratner, D. and Warmuth, M. (1986). Finding a shortest solution for the n × n extension of the 15-puzzle is intractable. In *AAAI-86*, Vol. 1, pp. 168–172.

Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, *3*(8), 1445–1450.

Rayward-Smith, V., Osman, I., Reeves, C., and Smith, G. (Eds.). (1996). *Modern Heuristic Search Methods*. Wiley.

Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. Library translation 1122, Royal Aircraft Establishment.

Reeson, C. G., Huang, K.-C., Bayer, K. M., and Choueiry, B. Y. (2007). An interactive constraint-based approach to sudoku. In *AAAI-07*, pp. 1976– 1977.

Regin, J. (1994). A filtering algorithm for constraints of difference in CSPs. In *AAAI-94*, pp. 362–367.

Reichenbach, H. (1949). *The Theory of Probability: An Inquiry into the Logical and Mathematical Foundations of the Calculus of Probability* (second edition). University of California Press.

Reid, D. B. (1979). An algorithm for tracking multiple targets. *IEEE Trans. Automatic Control*, *24*(6), 843–854.

Reif, J. (1979). Complexity of the mover's problem and generalizations. In *FOCS-79*, pp. 421–427. IEEE.

Reiter, R. (1980). A logic for default reasoning. *AIJ*, *13*(1–2), 81–132.

Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V. (Ed.), *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pp. 359–380. Academic Press.

Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Renner, G. and Ekart, A. (2003). Genetic algorithms in computer aided design. *Computer Aided Design*, *35*(8), 709–726.

Rényi, A. (1970). *Probability Theory*. Elsevier/North-Holland.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, *21*, 25–34. SIGGRAPH '87 Conference Proceedings.

Riazanov, A. and Voronkov, A. (2002). The design and implementation of VAMPIRE. *AI Communications*, *15*(2–3), 91–110.

Rich, E. and Knight, K. (1991). *Artificial Intelligence* (second edition). McGraw-Hill.

Richards, M. and Amir, E. (2007). Opponent modeling in Scrabble. In *IJCAI-07*.

Richardson, M., Bilmes, J., and Diorio, C. (2000). Hidden-articulator Markov models: Performance improvements and robustness to noise. In *ICASSP 00*.

Richter, S. and Westphal, M. (2008). The LAMA planner. In *Proc. International Planning Competition at ICAPS*.

Ridley, M. (2004). *Evolution*. Oxford Reader.

Rieger, C. (1976). An organization of knowledge for problem solving and language comprehension. *AIJ*, *7*, 89–127.

Riley, J. and Samuelson, W. (1981). Optimal auctions. *American Economic Review*, *71*, 381–392.

Riloff, E. (1993). Automatically constructing a dictionary for information extraction tasks. In *AAAI-93*, pp. 811–816.

Rintanen, J. (1999). Improvements to the evaluation of quantified Boolean formulae. In *IJCAI-99*, pp. 1192–1197.

Rintanen, J. (2007). Asymptotically optimal encodings of conformant planning in QBF. In *AAAI-07*,

pp. 1045–1050.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press.

Rissanen, J. (1984). Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, *IT-30*(4), 629–636.

Rissanen, J. (2007). *Information and Complexity in Statistical Modeling*. Springer.

Ritchie, G. D. and Hanna, F. K. (1984). AM: A case study in AI methodology. *AIJ*, *23*(3), 249–268.

Rivest, R. (1987). Learning decision lists. *Machine Learning*, *2*(3), 229–246.

Roberts, L. G. (1963). Machine perception of three-dimensional solids. Technical report 315, MIT Lincoln Laboratory.

Robertson, N. and Seymour, P. D. (1986). Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, *7*(3), 309–322.

Robertson, S. E. (1977). The probability ranking principle in IR. *J. Documentation*, *33*, 294–304.

Robertson, S. E. and Sparck Jones, K. (1976). Relevance weighting of search terms. *J. American Society for Information Science*, *27*, 129–146.

Robinson, A. and Voronkov, A. (2001). *Handbook of Automated Reasoning*. Elsevier.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *JACM*, *12*, 23–41.

Roche, E. and Schabes, Y. (1997). *Finite-State Language Processing (Language, Speech and Communication)*. Bradford Books.

Rock, I. (1984). *Perception*. W. H. Freeman.

Rosenblatt, F. (1957). The perceptron: A perceiving and recognizing automaton. Report 85-460-1, Project PARA, Cornell Aeronautical Laboratory.

Rosenblatt, F. (1960). On the convergence of reinforcement procedures in simple perceptrons. Report VG-1196-G-4, Cornell Aeronautical Laboratory.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan.

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, *27*, 832–837.

Rosenblueth, A., Wiener, N., and Bigelow, J. (1943). Behavior, purpose, and teleology. *Philosophy of Science*, *10*, 18–24.

Rosenschein, J. S. and Zlotkin, G. (1994). *Rules of Encounter*. MIT Press.

Rosenschein, S. J. (1985). Formal theories of knowledge in AI and robotics. *New Generation Computing*, *3*(4), 345–357.

Ross, P. E. (2004). Psyching out computer chess players. *IEEE Spectrum*, *41*(2), 14–15.

Ross, S. M. (1988). *A First Course in Probability* (third edition). Macmillan.

Rossi, F., van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Processing*. Elsevier.

Roussel, P. (1975). Prolog: Manual de reference et d'utilization. Tech. rep., Groupe d'Intelligence Artificielle, Universit´e d'Aix-Marseille.

Rouveirol, C. and Puget, J.-F. (1989). A simple and general solution for inverting resolution. In *Proc. European Working Session on Learning*, pp. 201–210.

Rowat, P. F. (1979). *Representing the Spatial Experience and Solving Spatial problems in a Simulated Robot Environment*. Ph.D. thesis, University of British Columbia.

Roweis, S. T. and Ghahramani, Z. (1999). A unifying review of Linear Gaussian Models. *Neural Computation*, *11*(2), 305–345.

Rowley, H., Baluja, S., and Kanade, T. (1996). Neural network-based face detection. In *CVPR*, pp. 203–208.

Roy, N., Gordon, G., and Thrun, S. (2005). Finding approximate POMDP solutions through belief compression. *JAIR*, *23*, 1–40.

Rubin, D. (1988). Using the SIR algorithm to simulate posterior distributions. In Bernardo, J. M., de Groot, M. H., Lindley, D. V., and Smith, A. F. M. (Eds.), *Bayesian Statistics 3*, pp. 395–402. Oxford University Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing*, Vol. 1, chap. 8, pp. 318–362. MIT Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Rumelhart, D. E. and McClelland, J. L. (Eds.). (1986). *Parallel Distributed Processing*. MIT Press.

Rummery, G. A. and Niranjan, M. (1994). Online Q-learning using connectionist systems. Tech. rep. CUED/F-INFENG/TR 166, Cambridge University Engineering Department.

Ruspini, E. H., Lowrance, J. D., and Strat, T. M. (1992). Understanding evidential reasoning. *IJAR*, *6*(3), 401–424.

Russell, J. G. B. (1990). Is screening for abdominal aortic aneurysm worthwhile? *Clinical Radiology*, *41*, 182–184.

Russell, S. J. (1985). The compleat guide to MRS. Report STAN-CS-85-1080, Computer Science Department, Stanford University.

Russell, S. J. (1986). A quantitative analysis of analogy by similarity. In *AAAI-86*, pp. 284–288.

Russell, S. J. (1988). Tree-structured bias. In *AAAI88*, Vol. 2, pp. 641–645.

Russell, S. J. (1992). Efficient memory-bounded search methods. In *ECAI-92*, pp. 1–5.

Russell, S. J. (1998). Learning agents for uncertain environments (extended abstract). In *COLT-98*,

pp. 101–103.

Russell, S. J., Binder, J., Koller, D., and Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *IJCAI-95*, pp. 1146–52.

Russell, S. J. and Grosof, B. (1987). A declarative approach to bias in concept learning. In *AAAI-87*.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd edition). Prentice-Hall.

Russell, S. J. and Subramanian, D. (1995). Provably bounded-optimal agents. *JAIR*, *3*, 575–609.

Russell, S. J., Subramanian, D., and Parr, R. (1993). Provably bounded optimal agents. In *IJCAI-93*, pp. 338–345.

Russell, S. J. and Wefald, E. H. (1989). On optimal game-tree search using rational meta-reasoning. In *IJCAI-89*, pp. 334–340.

Russell, S. J. and Wefald, E. H. (1991). *Do the Right Thing: Studies in Limited Rationality*. MIT Press.

Russell, S. J. and Wolfe, J. (2005). Efficient belief-state AND-OR search, with applications to Kriegspiel. In *IJCAI-05*, pp. 278–285.

Russell, S. J. and Zimdars, A. (2003). Q-decomposition of reinforcement learning agents. In *ICML-03*.

Rustagi, J. S. (1976). *Variational Methods in Statistics*. Academic Press.

Sabin, D. and Freuder, E. C. (1994). Contradicting conventional wisdom in constraint satisfaction. In *ECAI-94*, pp. 125–129.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *AIJ*, *5*(2), 115–135.

Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *IJCAI-75*, pp. 206–214.

Sacerdoti, E. D. (1977). *A Structure for Plans and Behavior*. Elsevier/North-Holland.

Sadri, F. and Kowalski, R. (1995). Variants of the event calculus. In *ICLP-95*, pp. 67–81.

Sahami, M., Dumais, S. T., Heckerman, D., and Horvitz, E. J. (1998). A Bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*.

Sahami, M., Hearst, M. A., and Saund, E. (1996). Applying the multiple cause mixture model to text categorization. In *ICML-96*, pp. 435–443.

Sahin, N. T., Pinker, S., Cash, S. S., Schomer, D., and Halgren, E. (2009). Sequential processing of lexical, grammatical, and phonological information within Broca's area. *Science*, *326*(5291), 445–449.

Sakuta, M. and Iida, H. (2002). AND/OR-tree search for solving problems with uncertainty: A case study using screen-shogi problems. *IPSJ Journal*, *43*(01).

Salomaa, A. (1969). Probabilistic and weighted grammars. *Information and Control*, *15*, 529–544.

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *CACM*, *18*(11), 613–620.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 210–229.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers II—Recent progress. *IBM Journal of Research and Development*, *11*(6), 601–617.

Samuelsson, C. and Rayner, M. (1991). Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *IJCAI-91*, pp. 609–615.

Sarawagi, S. (2007). Information extraction. *Foundations and Trends in Databases*, *1*(3), 261–377.

Satia, J. K. and Lave, R. E. (1973). Markovian decision processes with probabilistic observation of states. *Management Science*, *20*(1), 1–13.

Sato, T. and Kameya, Y. (1997). PRISM: A symbolic-statistical modeling language. In *IJCAI97*, pp. 1330–1335.

Saul, L. K., Jaakkola, T., and Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *JAIR*, *4*, 61–76.

Savage, L. J. (1954). *The Foundations of Statistics*. Wiley.

Sayre, K. (1993). Three more flaws in the computational model. Paper presented at the APA (Central Division) Annual Conference, Chicago, Illinois.

Schaeffer, J. (2008). *One Jump Ahead: Computer Perfection at Checkers*. Springer-Verlag.

Schaeffer, J., Burch, N., Bjornsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers is solved. *Science*, *317*, 1518– 1522.

Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum Associates.

Schank, R. C. and Riesbeck, C. (1981). *Inside Computer Understanding: Five Programs Plus Miniatures*. Lawrence Erlbaum Associates.

Schapire, R. E. and Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Machine Learning*, *39*(2/3), 135–168.

Schapire, R. E. (1990). The strength of weak learn-ability. *Machine Learning*, *5*(2), 197–227.

Schapire, R. E. (2003). The boosting approach to machine learning: An overview. In Denison, D. D., Hansen, M. H., Holmes, C., Mallick, B., and Yu, B. (Eds.), *Nonlinear Estimation and Classification*. Springer.

Schmid, C. and Mohr, R. (1996). Combining grey-value invariants with local constraints for object recognition. In *CVPR*.

Schmolze, J. G. and Lipkis, T. A. (1983). Classification in the KL-ONE representation system. In *IJCAI-83*, pp. 330–332.

Schöolkopf, B. and Smola, A. J. (2002). *Learning with Kernels*. MIT Press.

Schöoning, T. (1999). A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *FOCS99*, pp. 410–414.

Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. In *IJCAI87*, pp. 1039–1046.

Schoppers, M. J. (1989). In defense of reaction plans as caches. *AIMag*, *10*(4), 51–60.

Schröder, E. (1877). *Der Operationskreis des Logikkalküls*. B. G. Teubner, Leipzig.

Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, *275*, 1593.

Schulz, D., Burgard, W., Fox, D., and Cremers, A. B. (2003). People tracking with mobile robots using sample-based joint probabilistic data association filters. *Int. J. Robotics Research*, *22*(2), 99–116.

Schulz, S. (2004). System Description: E 0.81. In *Proc. International Joint Conference on Automated Reasoning*, Vol. 3097 of *LNAI*, pp. 223–228.

Schütze, H. (1995). *Ambiguity in Language Learning: Computational and Cognitive Models*. Ph.D. thesis, Stanford University. Also published by CSLI Press, 1997.

Schwartz, J. T., Scharir, M., and Hopcroft, J. (1987). *Planning, Geometry and Complexity of Robot Motion*. Ablex Publishing Corporation.

Schwartz, S. P. (Ed.). (1977). *Naming, Necessity, and Natural Kinds*. Cornell University Press.

Scott, D. and Krauss, P. (1966). Assigning probabilities to logical formulas. In Hintikka, J. and Suppes, P. (Eds.), *Aspects of Inductive Logic*. North-Holland.

Searle, J. R. (1980). Minds, brains, and programs. *BBS*, *3*, 417–457.

Searle, J. R. (1984). *Minds, Brains and Science*. Harvard University Press.

Searle, J. R. (1990). Is the brain's mind a computer program? *Scientific American*, *262*, 26–31.

Searle, J. R. (1992). *The Rediscovery of the Mind*. MIT Press.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, *34*(1), 1–47.

Segaran, T. (2007). *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. O'Reilly.

Selman, B., Kautz, H., and Cohen, B. (1996). Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26*, pp. 521–532. American Mathematical Society.

Selman, B. and Levesque, H. J. (1993). The complexity of path-based defeasible inheritance. *AIJ*, *62*(2), 303–339.

Selman, B., Levesque, H. J., and Mitchell, D. (1992). A new method for solving hard satisfiability

problems. In *AAAI-92*, pp. 440–446.

Sha, F. and Pereira, F. (2003). Shallow parsing with conditional random fields. Technical report CIS TR MS-CIS-02-35, Univ. of Penn.

Shachter, R. D. (1986). Evaluating influence diagrams. *Operations Research*, *34*, 871–882.

Shachter, R. D. (1998). Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *UAI-98*, pp. 480–487.

Shachter, R. D., D'Ambrosio, B., and Del Favero, B. A. (1990). Symbolic probabilistic inference in belief networks. In *AAAI-90*, pp. 126–131.

Shachter, R. D. and Kenley, C. R. (1989). Gaussian influence diagrams. *Management Science*, *35*(5), 527–550.

Shachter, R. D. and Peot, M. (1989). Simulation approaches to general probabilistic inference on belief networks. In *UAI-98*.

Shachter, R. D. and Heckerman, D. (1987). Thinking backward for knowledge acquisition. *AIMag*, *3*(Fall).

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

Shahookar, K. and Mazumder, P. (1991). VLSI cell placement techniques. *Computing Surveys*, *23*(2), 143–220.

Shanahan, M. (1997). *Solving the Frame Problem*. MIT Press.

Shanahan, M. (1999). The event calculus explained. In Wooldridge, M. J. and Veloso, M. (Eds.), *Artificial Intelligence Today*, pp. 409–430. Springer-Verlag.

Shankar, N. (1986). *Proof-Checking Metamathematics*. Ph.D. thesis, Computer Science Department, University of Texas at Austin.

Shannon, C. E. and Weaver, W. (1949). *The Mathematical Theory of Communication*. University of Illinois Press.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell Systems Technical Journal*, *27*, 379–423, 623–656.

Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine*, *41*(4), 256–275.

Shaparau, D., Pistore, M., and Traverso, P. (2008). Fusing procedural and declarative planning goals for nondeterministic domains. In *AAAI-08*.

Shapiro, E. (1981). An algorithm that infers theories from facts. In *IJCAI-81*, p. 1064.

Shapiro, S. C. (Ed.). (1992). *Encyclopedia of Artificial Intelligence* (second edition). Wiley.

Shapley, S. (1953). Stochastic games. In *PNAS*, Vol. 39, pp. 1095–1100.

Shatkay, H. and Kaelbling, L. P. (1997). Learning topological maps with weak local odometric information. In *IJCAI-97*.

Shelley, M. (1818). *Frankenstein: Or, the Modern Prometheus*. Pickering and Chatto.

Sheppard, B. (2002). World-championship-caliber scrabble. *AIJ*, *134*(1–2), 241–275.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *PAMI*, *22*(8), 888–905.

Shieber, S. (1994). Lessons from a restricted Turing Test. *CACM*, *37*, 70–78.

Shieber, S. (Ed.). (2004). *The Turing Test*. MIT Press.

Shoham, Y. (1993). Agent-oriented programming. *AIJ*, *60*(1), 51–92.

Shoham, Y. (1994). *Artificial Intelligence Techniques in Prolog*. Morgan Kaufmann.

Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge Univ. Press.

Shoham, Y., Powers, R., and Grenager, T. (2004). If multi-agent learning is the answer, what is the question? In *Proc. AAAI Fall Symposium on Artificial Multi-Agent Learning*.

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN*. Elsevier/North-Holland.

Sietsma, J. and Dow, R. J. F. (1988). Neural net pruning—Why and how. In *IEEE International Conference on Neural Networks*, pp. 325–333.

Siklossy, L. and Dreussi, J. (1973). An efficient robot planner which generates its own procedures. In *IJCAI-73*, pp. 423–430.

Silverstein, C., Henzinger, M., Marais, H., and Moricz, M. (1998). Analysis of a very large altavista query log. Tech. rep. 1998-014, Digital Systems Research Center.

Simmons, R. and Koenig, S. (1995). Probabilistic robot navigation in partially observable environments. In *IJCAI-95*, pp. 1080–1087. IJCAI, Inc.

Simon, D. (2006). *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley.

Simon, H. A. (1947). *Administrative behavior*. Macmillan.

Simon, H. A. (1957). *Models of Man: Social and Rational*. John Wiley.

Simon, H. A. (1963). Experiments with a heuristic compiler. *JACM*, *10*, 493–506.

Simon, H. A. (1981). *The Sciences of the Artificial* (second edition). MIT Press.

Simon, H. A. (1982). *Models of Bounded Rationality, Volume 1*. The MIT Press.

Simon, H. A. and Newell, A. (1958). Heuristic problem solving: The next advance in operations research. *Operations Research*, *6*, 1–10.

Simon, H. A. and Newell, A. (1961). Computer simulation of human thinking and problem solving. *Datamation*, *June/July*, 35–37.

Simon, J. C. and Dubois, O. (1989). Number of solutions to satisfiability instances—Applications to knowledge bases. *AIJ*, *3*, 53–65.

Simonis, H. (2005). Sudoku as a constraint problem. In *CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems*, pp. 13–27.

Singer, P. W. (2009). *Wired for War*. Penguin Press.

Singh, P., Lin, T., Mueller, E. T., Lim, G., Perkins, T., and Zhu, W. L. (2002). Open mind common sense: Knowledge acquisition from the general public. In *Proc. First International Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems*.

Singhal, A., Buckley, C., and Mitra, M. (1996). Pivoted document length normalization. In *SIGIR-96*, pp. 21–29.

Sittler, R. W. (1964). An optimal data association problem in surveillance theory. *IEEE Transactions on Military Electronics*, *8*(2), 125–139.

Skinner, B. F. (1953). *Science and Human Behavior*. Macmillan.

Skolem, T. (1920). Logisch-kombinatorische Untersuchungen über die Erfüllbarkeitoder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über die dichte Mengen. *Videnskapsselskapets skrifter, I. Matematisk-naturvidenskabelig klasse*, *4*.

Skolem, T. (1928). Über die mathematische Logik. *Norsk matematisk tidsskrift*, *10*, 125–142.

Slagle, J. R. (1963). A heuristic program that solves symbolic integration problems in freshman calculus. *JACM*, *10*(4).

Slate, D. J. and Atkin, L. R. (1977). CHESS 4.5— Northwestern University chess program. In Frey, P. W. (Ed.), *Chess Skill in Man and Machine*, pp. 82–118. Springer-Verlag.

Slater, E. (1950). Statistics for the chess computer and the factor of mobility. In *Symposium on Information Theory*, pp. 150–152. Ministry of Supply.

Sleator, D. and Temperley, D. (1993). Parsing English with a link grammar. In *Third Annual Workshop on Parsing technologies*.

Slocum, J. and Sonneveld, D. (2006). *The 15 Puzzle*. Slocum Puzzle Foundation.

Sloman, A. (1978). *The Computer Revolution in Philosophy*. Harvester Press.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, *21*, 1071–1088.

Smart, J. J. C. (1959). Sensations and brain processes. *Philosophical Review*, *68*, 141–156.

Smith, B. (2004). Ontology. In Floridi, L. (Ed.), *The Blackwell Guide to the Philosophy of Computing and Information*, pp. 155–166. Wiley-Blackwell.

Smith, D. E., Genesereth, M. R., and Ginsberg, M. L. (1986). Controlling recursive inference. *AIJ*, *30*(3), 343–389.

Smith, D. A. and Eisner, J. (2008). Dependency parsing by belief propagation. In *EMNLP*, pp. 145–156.

Smith, D. E. and Weld, D. S. (1998). Conformant Graphplan. In *AAAI-98*, pp. 889–896.

Smith, J. Q. (1988). *Decision Analysis*. Chapman and Hall.

Smith, J. E. and Winkler, R. L. (2006). The optimizer’s curse: Skepticism and postdecision surprise in decision analysis. *Management Science*, *52*(3), 311–322.

Smith, J. M. (1982). *Evolution and the Theory of Games*. Cambridge University Press.

Smith, J. M. and Szathm′ary, E. (1999). *The Origins of Life: From the Birth of Life to the Origin of Language*. Oxford University Press.

Smith, M. K., Welty, C., and McGuinness, D. (2004). OWL web ontology language guide. Tech. rep., W3C.

Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *Int. J. Robotics Research*, *5*(4), 56–68.

Smith, S. J. J., Nau, D. S., and Throop, T. A. (1998). Success in spades: Using AI planning techniques to win the world championship of computer bridge. In *AAAI-98*, pp. 1079–1086.

Smolensky, P. (1988). On the proper treatment of connectionism. *BBS*, *2*, 1–74.

Smullyan, R. M. (1995). *First-Order Logic*. Dover.

Smyth, P., Heckerman, D., and Jordan, M. I. (1997). Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, *9*(2), 227–269.

Snell, M. B. (2008). Do you have free will? John Searle reflects on various philosophical questions in light of new research on the brain. *California Alumni Magazine*, *March/April*.

Soderland, S. and Weld, D. S. (1991). Evaluating nonlinear planning. Technical report TR-91-02-03, University of Washington Department of Computer Science and Engineering.

Solomonoff, R. J. (1964). A formal theory of inductive inference. *Information and Control*, *7*, 1–22, 224–254.

Solomonoff, R. J. (2009). Algorithmic probability– theory and applications. In Emmert-Streib, F. and Dehmer, M. (Eds.), *Information Theory and Statitical Learning*. Springer.

Sondik, E. J. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. Ph.D. thesis, Stanford University.

Sosic, R. and Gu, J. (1994). Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering*, *6*(5), 661–668.

Sowa, J. (1999). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Blackwell.

Spaan, M. T. J. and Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *JAIR*, *24*, 195–220.

Spiegelhalter, D. J., Dawid, A. P., Lauritzen, S., and Cowell, R. (1993). Bayesian analysis in expert systems. *Statistical Science*, *8*, 219–282.

Spielberg, S. (2001). AI. Movie.

Spirtes, P., Glymour, C., and Scheines, R. (1993). *Causation, prediction, and search*. Springer-Verlag.

Srinivasan, A., Muggleton, S. H., King, R. D., and Sternberg, M. J. E. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. In *ILP-94*, Vol. 237, pp. 217–232.

Srivas, M. and Bickford, M. (1990). Formal verification of a pipelined microprocessor. *IEEE Software*, *7*(5), 52–64.

Staab, S. (2004). *Handbook on Ontologies*. Springer.

Stallman, R. M. and Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *AIJ*, *9*(2), 135–196.

Stanfill, C. and Waltz, D. (1986). Toward memory-based reasoning. *CACM*, *29*(12), 1213–1228.

Stefik, M. (1995). *Introduction to Knowledge Systems*. Morgan Kaufmann.

Stein, L. A. (2002). *Interactive Programming in Java (pre-publication draft)*. Morgan Kaufmann.

Stephenson, T., Bourlard, H., Bengio, S., and Morris, A. (2000). Automatic speech recognition using dynamic bayesian networks with both acoustic and articulatory features. In *ICSLP-00*, pp. 951–954.

Stergiou, K. and Walsh, T. (1999). The difference all-difference makes. In *IJCAI-99*, pp. 414–419.

Stickel, M. E. (1992). A prolog technology theorem prover: a new exposition and implementation in prolog. *Theoretical Computer Science*, *104*, 109–128.

Stiller, L. (1992). KQNKRR. *J. International Computer Chess Association*, *15*(1), 16–18.

Stiller, L. (1996). Multilinear algebra and chess endgames. In Nowakowski, R. J. (Ed.), *Games of No Chance, MSRI, 29, 1996*. Mathematical Sciences Research Institute.

Stockman, G. (1979). A minimax algorithm better than alpha–beta? *AIJ*, *12*(2), 179–196.

Stoffel, K., Taylor, M., and Hendler, J. (1997). Efficient management of very large ontologies. In *Proc. AAAI-97*, pp. 442–447.

Stolcke, A. and Omohundro, S. (1994). Inducing probabilistic grammars by Bayesian model merging. In *Proc. Second International Colloquium on Grammatical Inference and Applications (ICGI-94)*, pp. 106–118.

Stone, M. (1974). Cross-validatory choice and assessment of statostical predictions. *J. Royal Statistical Society*, *36*(111–133).

Stone, P. (2000). *Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer*. MIT Press.

Stone, P. (2003). Multiagent competitions and research: Lessons from RoboCup and TAC. In Lima, P. U. and Rojas, P. (Eds.), *RoboCup-2002: Robot Soccer World Cup VI*, pp. 224–237. Springer Verlag.

Stone, P., Kaminka, G., and Rosenschein, J. S. (2009). Leading a best-response teammate in an ad hoc team. In *AAMAS Workshop in Agent Mediated Electronic Commerce*.

Stork, D. G. (2004). Optics and realism in rennaissance art. *Scientific American*, pp. 77–83.

Strachey, C. (1952). Logical or non-mathematical programmes. In *Proc. 1952 ACM national meeting (Toronto)*, pp. 46–49.

Stratonovich, R. L. (1959). Optimum nonlinear systems which bring about a separation of a signal with constant parameters from noise. *Radiofizika*, *2*(6), 892–901.

Stratonovich, R. L. (1965). On value of information. *Izvestiya of USSR Academy of Sciences, Technical Cybernetics*, *5*, 3–12.

Subramanian, D. and Feldman, R. (1990). The utility of EBL in recursive domain theories. In *AAAI-90*, Vol. 2, pp. 942–949.

Subramanian, D. and Wang, E. (1994). Constraint-based kinematic synthesis. In *Proc. International Conference on Qualitative Reasoning*, pp. 228–239.

Sussman, G. J. (1975). *A Computer Model of Skill Acquisition*. Elsevier/North-Holland.

Sutcliffe, G. and Suttner, C. (1998). The TPTP Problem Library: CNF Release v1.2.1. *JAR*, *21*(2), 177–203.

Sutcliffe, G., Schulz, S., Claessen, K., and Gelder, A. V. (2006). Using the TPTP language for writing derivations and finite interpretations. In *Proc. International Joint Conference on Automated Reasoning*, pp. 67–81.

Sutherland, I. (1963). Sketchpad: A man-machine graphical communication system. In *Proc. Spring Joint Computer Conference*, pp. 329–346.

Sutton, C. and McCallum, A. (2007). An introduction to conditional random fields for relational learning. In Getoor, L. and Taskar, B. (Eds.), *Introduction to Statistical Relational Learning*. MIT Press.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T.K., and Müller, K.-R. (Eds.), *NIPS 12*, pp. 1057–1063. MIT Press.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML-90*, pp. 216–224.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Svore, K. and Burges, C. (2009). A machine learning approach for improved bm25 retrieval. In *Proc. Conference on Information Knowledge Management*.

Swade, D. (2000). *Difference Engine: Charles Babbage And The Quest To Build The First Computer*. Diane Publishing Co.

Swerling, P. (1959). First order error propagation in a stagewise smoothing procedure for satellite observations. *J. Astronautical Sciences*, *6*, 46–52.

Swift, T. and Warren, D. S. (1994). Analysis of SLGWAM evaluation of definite programs. In *Logic Programming. Proc. 1994 International Symposium on Logic programming*, pp. 219–235.

Syrjänen, T. (2000). Lparse 1.0 user's manual. saturn.tcs.hut.fi/Software/smodels.

Tadepalli, P. (1993). Learning from queries and examples with tree-structured bias. In *ICML-93*, pp. 322–329.

Tadepalli, P., Givan, R., and Driessens, K. (2004). Relational reinforcement learning: An overview. In *ICML-04*.

Tait, P. G. (1880). Note on the theory of the "15 puzzle". *Proc. Royal Society of Edinburgh*, *10*, 664–665.

Tamaki, H. and Sato, T. (1986). OLD resolution with tabulation. In *ICLP-86*, pp. 84–98.

Tarjan, R. E. (1983). *Data Structures and Network Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM (Society for Industrial and Applied Mathematics).

Tarski, A. (1935). Die Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, *1*, 261–405.

Tarski, A. (1941). *Introduction to Logic and to the Methodology of Deductive Sciences*. Dover.

Tarski, A. (1956). *Logic, Semantics, Metamathematics: Papers from 1923 to 1938*. Oxford University Press.

Tash, J. K. and Russell, S. J. (1994). Control strategies for a stochastic planner. In *AAAI-94*, pp. 1079– 1085.

Taskar, B., Abbeel, P., and Koller, D. (2002). Discriminative probabilistic models for relational data. In *UAI-02*.

Tate, A. (1975a). Interacting goals and their use. In *IJCAI-75*, pp. 215–218.

Tate, A. (1975b). *Using Goal Structure to Direct Search in a Problem Solver*. Ph.D. thesis, University of Edinburgh.

Tate, A. (1977). Generating project networks. In *IJCAI-77*, pp. 888–893.

Tate, A. and Whiter, A. M. (1984). Planning with multiple resource constraints and an application to a naval planning problem. In *Proc. First Conference on AI Applications*, pp. 410–416.

Tatman, J. A. and Shachter, R. D. (1990). Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, *20*(2), 365–379.

Tattersall, C. (1911). *A Thousand End-Games: A Collection of Chess Positions That Can be Won or Drawn by the Best Play*. British Chess Magazine.

Taylor, G., Stensrud, B., Eitelman, S., and Dunham, C. (2007). Towards automating airspace management. In *Proc. Computational Intelligence for Security and Defense Applications (CISDA) Conference*, pp. 1–5.

Tenenbaum, J., Griffiths, T., and Niyogi, S. (2007). Intuitive theories as grammars for causal

inference. In Gopnik, A. and Schulz, L. (Eds.), *Causal learning: Psychology, Philosophy, and Computation*. Oxford University Press.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*(3–4), 257–277.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *CACM*, *38*(3), 58–68.

Tesauro, G. and Sejnowski, T. (1989). A parallel network that learns to play backgammon. *AIJ*, *39*(3), 357–390.

Teyssier, M. and Koller, D. (2005). Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *UAI-05*, pp. 584–590.

Thaler, R. (1992). *The Winner's Curse: Paradoxes and Anomalies of Economic Life*. Princeton University Press.

Thaler, R. and Sunstein, C. (2009). *Nudge: Improving Decisions About Health, Wealth, and Happiness*. Penguin.

Theocharous, G., Murphy, K., and Kaelbling, L. P. (2004). Representing hierarchical POMDPs as DBNs for multi-scale robot localization. In *ICRA 04*.

Thiele, T. (1880). Om anvendelse af mindste kvadraters methode i nogle tilfælde, hvor en komplikation af visse slags uensartede tilfældige fejlkilder giver fejlene en 'systematisk' karakter. *Vidensk. Selsk. Skr. 5. Rk., naturvid. og mat. Afd.*, *12*, 381–408.

Thielscher, M. (1999). From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *AIJ*, *111*(1–2), 277–299.

Thompson, K. (1986). Retrograde analysis of certain endgames. *J. International Computer Chess Association*, *May*, 131–139.

Thompson, K. (1996). 6-piece endgames. *J. International Computer Chess Association*, *19*(4), 215–226.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

Thrun, S., Fox, D., and Burgard, W. (1998). A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, *31*, 29–53.

Thrun, S. (2006). Stanley, the robot that won the DARPA Grand Challenge. *J. Field Robotics*, *23*(9), 661–692.

Tikhonov, A. N. (1963). Solution of incorrectly formulated problems and the regularization method. *Soviet Math. Dokl.*, *5*, 1035–1038.

Titterington, D. M., Smith, A. F. M., and Makov, U. E. (1985). *Statistical analysis of finite mixture distributions*. Wiley.

Toffler, A. (1970). *Future Shock*. Bantam.

Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography: A factorization method. *IJCV*, *9*, 137–154.

Torralba, A., Fergus, R., and Weiss, Y. (2008). Small codes and large image databases for recognition. In *CVPR*, pp. 1–8.

Trucco, E. and Verri, A. (1998). *Introductory Techniques for 3-D Computer Vision*. Prentice Hall.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*(5), 674–690.

Tumer, K. and Wolpert, D. (2000). Collective intelligence and braess' paradox. In *AAAI-00*, pp. 104–109.

Turcotte, M., Muggleton, S. H., and Sternberg, M. J. E. (2001). Automated discovery of structural signatures of protein fold and function. *J. Molecular Biology*, *306*, 591–605.

Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Mathematical Society, 2nd series*, *42*, 230–265.

Turing, A. (1948). Intelligent machinery. Tech. rep., National Physical Laboratory. reprinted in (Ince, 1992).

Turing, A. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–460.

Turing, A., Strachey, C., Bates, M. A., and Bowden, B. V. (1953). Digital computers applied to games. In Bowden, B. V. (Ed.), *Faster than Thought*, pp. 286–310. Pitman.

Tversky, A. and Kahneman, D. (1982). Causal schemata in judgements under uncertainty. In Kahneman, D., Slovic, P., and Tversky, A. (Eds.), *Judgement Under Uncertainty: Heuristics and Biases*. Cambridge University Press.

Ullman, J. D. (1985). Implementation of logical query languages for databases. *ACM Transactions on Database Systems*, *10*(3), 289–321.

Ullman, S. (1979). *The Interpretation of Visual Motion*. MIT Press.

Urmson, C. and Whittaker, W. (2008). Self-driving cars and the Urban Challenge. *IEEE Intelligent Systems*, *23*(2), 66–68.

Valiant, L. (1984). A theory of the learnable. *CACM*, *27*, 1134–1142.

van Beek, P. (2006). Backtracking search algorithms. In Rossi, F., van Beek, P., and Walsh, T. (Eds.), *Handbook of Constraint Programming*. Else-vier.

van Beek, P. and Chen, X. (1999). CPlan: A constraint programming approach to planning. In *AAAI99*, pp. 585–590.

van Beek, P. and Manchak, D. (1996). The design and experimental analysis of algorithms for temporal reasoning. *JAIR*, *4*, 1–18.

van Bentham, J. and ter Meulen, A. (1997). *Handbook of Logic and Language*. MIT Press.

Van Emden, M. H. and Kowalski, R. (1976). The semantics of predicate logic as a programming language. *JACM*, *23*(4), 733–742.

van Harmelen, F. and Bundy, A. (1988). Explanation-based generalisation = partial evaluation. *AIJ*,

36(3), 401–412.

van Harmelen, F., Lifschitz, V., and Porter, B. (2007). *The Handbook of Knowledge Representation*. Elsevier.

van Heijenoort, J. (Ed.). (1967). *From Frege to G¨odel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press.

Van Hentenryck, P., Saraswat, V., and Deville, Y. (1998). Design, implementation, and evaluation of the constraint language cc(FD). *J. Logic Programming*, *37*(1–3), 139–164.

van Hoeve, W.-J. (2001). The alldifferent constraint: a survey. In *6th Annual Workshop of the ERCIM Working Group on Constraints*.

van Hoeve, W.-J. and Katriel, I. (2006). Global constraints. In Rossi, F., van Beek, P., and Walsh, T. (Eds.), *Handbook of Constraint Processing*, pp. 169–208. Elsevier.

van Lambalgen, M. and Hamm, F. (2005). *The Proper Treatment of Events*. Wiley-Blackwell.

van Nunen, J. A. E. E. (1976). A set of successive approximation methods for discounted Markovian decision problems. *Zeitschrift fur Operations Research, Serie A*, *20*(5), 203–208.

Van Roy, B. (1998). *Learning and value function approximation in complex decision processes*. Ph.D. thesis, Laboratory for Information and Decision Systems, MIT.

Van Roy, P. L. (1990). Can logic programming execute as fast as imperative programming? Report UCB/CSD 90/600, Computer Science Division, University of California, Berkeley, California.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley.

Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, *16*, 264–280.

Varian, H. R. (1995). Economic mechanism design for computerized agents. In *USENIX Workshop on Electronic Commerce*, pp. 13–21.

Vauquois, B. (1968). A survey of formal grammars and algorithms for recognition and transformation in mechanical translation. In *Proc. IFIP Congress*, pp. 1114–1122.

Veloso, M. and Carbonell, J. G. (1993). Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, *10*, 249–278.

Vere, S. A. (1983). Planning in time: Windows and durations for activities and goals. *PAMI*, *5*, 246–267.

Verma, V., Gordon, G., Simmons, R., and Thrun, S. (2004). Particle filters for rover fault diagnosis. *IEEE Robotics and Automation Magazine*, *June*.

Vinge, V. (1993). The coming technological singularity: How to survive in the post-human era. In *VISION-21 Symposium*. NASA Lewis Research Center and the Ohio Aerospace Institute.

Viola, P. and Jones, M. (2002a). Fast and robust classification using asymmetric adaboost and a detector cascade. In *NIPS 14*.

Viola, P. and Jones, M. (2002b). Robust real-time object detection. *ICCV*.

Visser, U. and Burkhard, H.-D. (2007). RoboCup 2006: achievements and goals for the future. *AIMag*, *28*(2), 115–130.

Visser, U., Ribeiro, F., Ohashi, T., and Dellaert, F. (Eds.). (2008). *RoboCup 2007: Robot Soccer World Cup XI*. Springer.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *13*(2), 260–269.

Vlassis, N. (2008). *A Concise Introduction to Multi-agent Systems and Distributed Artificial Intelligence*. Morgan and Claypool.

von Mises, R. (1928). *Wahrscheinlichkeit, Statistik und Wahrheit*. J. Springer.

von Neumann, J. (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, *100*(295–320).

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior* (first edition). Princeton University Press.

von Winterfeldt, D. and Edwards, W. (1986). *Decision Analysis and Behavioral Research*. Cambridge University Press.

Vossen, T., Ball, M., Lotem, A., and Nau, D. S. (2001). Applying integer programming to AI planning. *Knowledge Engineering Review*, *16*, 85–100.

Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Machine Learning*, *1*(1–2), 1–305.

Waldinger, R. (1975). Achieving several goals simultaneously. In Elcock, E. W. and Michie, D. (Eds.), *Machine Intelligence 8*, pp. 94–138. Ellis Horwood.

Wallace, A. R. (1858). On the tendency of varieties to depart indefinitely from the original type. *Proc. Linnean Society of London*, *3*, 53–62.

Waltz, D. (1975). Understanding line drawings of scenes with shadows. In Winston, P. H. (Ed.), *The Psychology of Computer Vision*. McGraw-Hill.

Wang, Y. and Gelly, S. (2007). Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *IEEE Symposium on Computational Intelligence and Games*, pp. 175–182.

Wanner, E. (1974). *On remembering, forgetting and understanding sentences*. Mouton.

Warren, D. H. D. (1974). WARPLAN: A System for Generating Plans. Department of Computational Logic Memo 76, University of Edinburgh.

Warren, D. H. D. (1983). An abstract Prolog instruction set. Technical note 309, SRI International.

Warren, D. H. D., Pereira, L. M., and Pereira, F. (1977). PROLOG: The language and its implementation compared with LISP. *SIGPLAN Notices*, *12*(8), 109–115.

Wasserman, L. (2004). *All of Statistics*. Springer.

Watkins, C. J. (1989). *Models of Delayed Reinforcement Learning*. Ph.D. thesis, Psychology Department, Cambridge University.

Watson, J. D. and Crick, F. H. C. (1953). A structure for deoxyribose nucleic acid. *Nature*, *171*, 737.

Waugh, K., Schnizlein, D., Bowling, M., and Szafron, D. (2009). Abstraction pathologies in extensive games. In *AAMAS-09*.

Weaver, W. (1949). Translation. In Locke, W. N. and Booth, D. (Eds.), *Machine translation of languages: fourteen essays*, pp. 15–23. Wiley.

Webber, B. L. and Nilsson, N. J. (Eds.). (1981). *Readings in Artificial Intelligence*. Morgan Kaufmann.

Weibull, J. (1995). *Evolutionary Game Theory*. MIT Press.

Weidenbach, C. (2001). SPASS: Combining superposition, sorts and splitting. In Robinson, A. and Voronkov, A. (Eds.), *Handbook of Automated Reasoning*. MIT Press.

Weiss, G. (2000a). *Multiagent systems*. MIT Press.

Weiss, Y. (2000b). Correctness of local probability propagation in graphical models with loops. *Neural Computation*, *12*(1), 1–41.

Weiss, Y. and Freeman, W. (2001). Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Computation*, *13*(10), 2173–2200.

Weizenbaum, J. (1976). *Computer Power and Human Reason*. W. H. Freeman.

Weld, D. S. (1994). An introduction to least commitment planning. *AIMag*, *15*(4), 27–61.

Weld, D. S. (1999). Recent advances in AI planning. *AIMag*, *20*(2), 93–122.

Weld, D. S., Anderson, C. R., and Smith, D. E. (1998). Extending graphplan to handle uncertainty and sensing actions. In *AAAI-98*, pp. 897–904.

Weld, D. S. and de Kleer, J. (1990). *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann.

Weld, D. S. and Etzioni, O. (1994). The first law of robotics: A call to arms. In *AAAI-94*.

Wellman, M. P. (1985). Reasoning about preference models. Technical report MIT/LCS/TR-340, Laboratory for Computer Science, MIT.

Wellman, M. P. (1988). *Formulation of Tradeoffs in Planning under Uncertainty*. Ph.D. thesis, Massachusetts Institute of Technology.

Wellman, M. P. (1990a). Fundamental concepts of qualitative probabilistic networks. *AIJ*, *44*(3), 257–303.

Wellman, M. P. (1990b). The STRIPS assumption for planning under uncertainty. In *AAAI-90*, pp. 198–203.

Wellman, M. P. (1995). The economic approach to artificial intelligence. *ACM Computing Surveys*,

27(3), 360–362.

Wellman, M. P., Breese, J. S., and Goldman, R. (1992). From knowledge bases to decision models. *Knowledge Engineering Review*, *7*(1), 35–53.

Wellman, M. P. and Doyle, J. (1992). Modular utility representation for decision-theoretic planning. In *ICAPS-92*, pp. 236–242.

Wellman, M. P., Wurman, P., O'Malley, K., Bangera, R., Lin, S., Reeves, D., and Walsh, W. (2001). A trading agent competition. *IEEE Internet Computing*.

Wells, H. G. (1898). *The War of the Worlds*. William Heinemann.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. thesis, Harvard University.

Werbos, P. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook*, *22*, 25–38.

Wesley, M. A. and Lozano-Perez, T. (1979). An algorithm for planning collision-free paths among polyhedral objects. *CACM*, *22*(10), 560–570.

Wexler, Y. and Meek, C. (2009). MAS: A multiplicative approximation scheme for probabilistic inference. In *NIPS 21*.

Whitehead, A. N. (1911). *An Introduction to Mathematics*. Williams and Northgate.

Whitehead, A. N. and Russell, B. (1910). *Principia Mathematica*. Cambridge University Press.

Whorf, B. (1956). *Language, Thought, and Reality*. MIT Press.

Widrow, B. (1962). Generalization and information storage in networks of adaline "neurons". In *Self-Organizing Systems 1962*, pp. 435–461.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pp. 96–104.

Wiedijk, F. (2003). Comparing mathematical provers. In *Mathematical Knowledge Management*, pp. 188–202.

Wiegley, J., Goldberg, K., Peshkin, M., and Brokowski, M. (1996). A complete algorithm for designing passive fences to orient parts. In *ICRA 96*.

Wiener, N. (1942). The extrapolation, interpolation, and smoothing of stationary time series. Osrd 370, Report to the Services 19, Research Project DIC6037, MIT.

Wiener, N. (1948). *Cybernetics*. Wiley.

Wilensky, R. (1978). *Understanding goal-based stories*. Ph.D. thesis, Yale University.

Wilensky, R. (1983). *Planning and Understanding*. Addison-Wesley.

Wilkins, D. E. (1980). Using patterns and plans in chess. *AIJ*, *14*(2), 165–203.

Wilkins, D. E. (1988). *Practical Planning: Extending the AI Planning Paradigm*. Morgan

Kaufmann.

Wilkins, D. E. (1990). Can AI planners solve practical problems? *Computational Intelligence*, *6*(4), 232–246.

Williams, B., Ingham, M., Chung, S., and Elliott, P. (2003). Model-based programming of intelligent embedded systems and robotic space explorers. In *Proc. IEEE: Special Issue on Modeling and Design of Embedded Software*, pp. 212–237.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.

Williams, R. J. and Baird, L. C. I. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Tech. rep. NU-CCS-93-14, College of Computer Science, Northeastern University.

Wilson, R. A. and Keil, F. C. (Eds.). (1999). *The MIT Encyclopedia of the Cognitive Sciences*. MIT Press.

Wilson, R. (2004). *Four Colors Suffice*. Princeton University Press.

Winograd, S. and Cowan, J. D. (1963). *Reliable Computation in the Presence of Noise*. MIT Press.

Winograd, T. (1972). Understanding natural language. *Cognitive Psychology*, *3*(1), 1–191.

Winston, P. H. (1970). Learning structural descriptions from examples. Technical report MAC-TR-76, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Winston, P. H. (1992). *Artificial Intelligence* (Third edition). Addison-Wesley.

Wintermute, S., Xu, J., and Laird, J. (2007). SORTS: A human-level approach to real-time strategy AI. In *Proc. Third Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE07)*.

Witten, I. H. and Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, *37*(4), 1085–1094.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques* (2nd edition). Morgan Kaufmann.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images* (second edition). Morgan Kaufmann.

Wittgenstein, L. (1922). *Tractatus Logico-Philosophicus* (second edition). Routledge and Kegan Paul. Reprinted 1971, edited by D. F. Pears and B. F. McGuinness. This edition of the English translation also contains Wittgenstein's original German text on facing pages, as well as Bertrand Russell's introduction to the 1922 edition.

Wittgenstein, L. (1953). *Philosophical Investigations*. Macmillan.

Wojciechowski, W. S. and Wojcik, A. S. (1983). Automated design of multiple-valued logic circuits

by automated theorem proving techniques. *IEEE Transactions on Computers*, *C-32*(9), 785–798.

Wolfe, J. and Russell, S. J. (2007). Exploiting belief state structure in graph search. In *ICAPS Workshop on Planning in Games*.

Woods, W. A. (1973). Progress in natural language understanding: An application to lunar geology. In *AFIPS Conference Proceedings*, Vol. 42, pp. 441–450.

Woods, W. A. (1975). What's in a link? Foundations for semantic networks. In Bobrow, D. G. and Collins, A. M. (Eds.), *Representation and Understanding: Studies in Cognitive Science*, pp. 35–82. Academic Press.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. Wiley.

Wooldridge, M. and Rao, A. (Eds.). (1999). *Foundations of rational agency*. Kluwer.

Wos, L., Carson, D., and Robinson, G. (1964). The unit preference strategy in theorem proving. In *Proc. Fall Joint Computer Conference*, pp. 615–621.

Wos, L., Carson, D., and Robinson, G. (1965). Efficiency and completeness of the set-of-support strategy in theorem proving. *JACM*, *12*, 536–541.

Wos, L., Overbeek, R., Lusk, E., and Boyle, J. (1992). *Automated Reasoning: Introduction and Applications* (second edition). McGraw-Hill.

Wos, L. and Robinson, G. (1968). Paramodulation and set of support. In *Proc. IRIA Symposium on Automatic Demonstration*, pp. 276–310.

Wos, L., Robinson, G., Carson, D., and Shalla, L. (1967). The concept of demodulation in theorem proving. *JACM*, *14*, 698–704.

Wos, L. and Winker, S. (1983). Open questions solved with the assistance of AURA. In *Automated Theorem Proving: After 25 Years: Proc. Special Session of the 89th Annual Meeting of the American Mathematical Society*, pp. 71–88. American Mathematical Society.

Wos, L. and Pieper, G. (2003). *Automated Reasoning and the Discovery of Missing and Elegant Proofs*. Rinton Press.

Wray, R. E. and Jones, R. M. (2005). An introduction to Soar as an agent architecture. In Sun, R. (Ed.), *Cognition and Multi-agent Interaction: From Cognitive Modeling to Social Simulation*, pp. 53–78. Cambridge University Press.

Wright, S. (1921). Correlation and causation. *J. Agricultural Research*, *20*, 557–585.

Wright, S. (1931). Evolution in Mendelian populations. *Genetics*, *16*, 97–159.

Wright, S. (1934). The method of path coefficients. *Annals of Mathematical Statistics*, *5*, 161–215.

Wu, D. (1993). Estimating probability distributions over hypotheses with variable unification. In *IJCAI93*, pp. 790–795.

Wu, F. and Weld, D. S. (2008). Automatically refining the wikipedia infobox ontology. In *17th World Wide Web Conference (WWW2008)*.

Yang, F., Culberson, J., Holte, R., Zahavi, U., and Felner, A. (2008). A general theory of additive state space abstractions. *JAIR*, *32*, 631–662.

Yang, Q. (1990). Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, *6*, 12–24.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *ACL95*, pp. 189–196.

Yedidia, J., Freeman, W., and Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, *51*(7), 2282–2312.

Yip, K. M.-K. (1991). *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer*. MIT Press.

Yngve, V. (1955). A model and an hypothesis for language structure. In Locke, W. N. and Booth, A. D. (Eds.), *Machine Translation of Languages*, pp. 208–226. MIT Press.

Yob, G. (1975). Hunt the wumpus! *Creative Computing*, *Sep/Oct*.

Yoshikawa, T. (1990). *Foundations of Robotics: Analysis and Control*. MIT Press.

Young, H. P. (2004). *Strategic Learning and Its Limits*. Oxford University Press.

Younger, D. H. (1967). Recognition and parsing of 3 context-free languages in time n. *Information and Control*, *10*(2), 189–208.

Yudkowsky, E. (2008). Artificial intelligence as a positive and negative factor in global risk. In Bostrom, N. and Cirkovic, M. (Eds.), *Global Catastrophic Risk*. Oxford University Press.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.

Zadeh, L. A. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, *1*, 3–28.

Zaritskii, V. S., Svetnik, V. B., and Shimelevich, L. I. (1975). Monte-Carlo technique in problems of optimal information processing. *Automation and Remote Control*, *36*, 2015–22.

Zelle, J. and Mooney, R. (1996). Learning to parse database queries using inductive logic programming. In *AAAI-96*, pp. 1050–1055.

Zermelo, E. (1913). Uber Eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In *Proc. Fifth International Congress of Mathematicians*, Vol. 2, pp. 501–504.

Zermelo, E. (1976). An application of set theory to the theory of chess-playing. *Firbush News*, *6*, 37–42. English translation of (Zermelo 1913).

Zettlemoyer, L. S. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI-05*.

Zhang, H. and Stickel, M. E. (1996). An efficient algorithm for unit-propagation. In *Proc. Fourth International Symposium on Artificial Intelligence and Mathematics*.

Zhang, L., Pavlovic, V., Cantor, C. R., and Kasif, S. (2003). Human-mouse gene identification by comparative evidence integration and evolutionary analysis. *Genome Research*, pp. 1–13.

Zhang, N. L. and Poole, D. (1994). A simple approach to Bayesian network computations. In *Proc. 10th Canadian Conference on Artificial Intelligence*, pp. 171–178.

Zhang, N. L., Qi, R., and Poole, D. (1994). A computational theory of decision networks. *IJAR*, *11*, 83–158.

Zhou, R. and Hansen, E. (2002). Memory-bounded A* graph search. In *Proc. 15th International Flairs Conference*.

Zhou, R. and Hansen, E. (2006). Breadth-first heuristic search. *AIJ*, *170*(4–5), 385–408.

Zhu, D. J. and Latombe, J.-C. (1991). New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, *7*(1), 9–20.

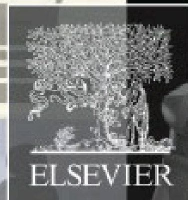
Zimmermann, H.-J. (Ed.). (1999). *Practical applications of fuzzy technologies*. Kluwer.

Zimmermann, H.-J. (2001). *Fuzzy Set Theory— And Its Applications* (Fourth edition). Kluwer.

Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C. (2008). Regret minimization in games with incomplete information. In *NIPS 20*, pp. 1729–1736.

Zollmann, A., Venugopal, A., Och, F. J., and Ponte, J. (2008). A systematic comparison of phrase-based, hierarchical and syntax-augmented statistical MT. In *COLING-08*.

Zweig, G. and Russell, S. J. (1998). Speech recognition with dynamic Bayesian networks. In *AAAI-98*, pp. 173–180.

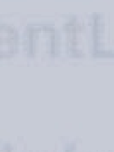




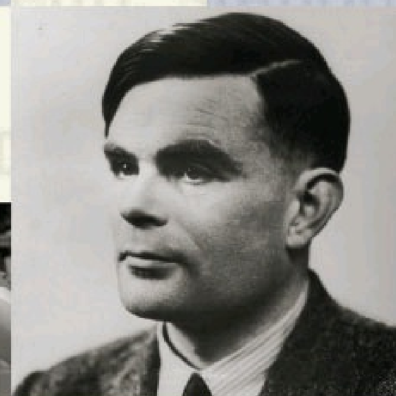








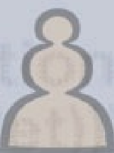





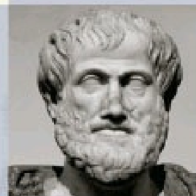













Stuart
Russell
Peter
Norvig



Inteligência Artificial

Tradução da Terceira Edição

Referência Completa para Cursos de Computação

Adotado em mais de 750 Universidades em 85 Países